

普通高等教育“十一五”国家级规划教材



21世纪大学本科计算机专业系列教材

袁春风 编著

计算机组成与系统结构习题解答与教学指导

http://www.tup.com.cn

- 根据教育部“高等学校计算机科学与技术专业规范”组织编写
- 与美国 ACM 和 IEEE CS *Computing Curricula* 最新进展同步
- 教育部–微软精品课程配套教材
- 网络教育国家精品课程配套教材

清华大学出版社

普通高等教育“十一五”国家级规划教材
21 世纪大学本科计算机专业系列教材

计算机组成与系统结构
习题解答与教学指导

袁春风　编著

清华大学出版社
北京

内 容 简 介

本书作为《计算机组成与系统结构》教材的教学指导用书，主要对每个章节的教学目标和内容安排、主要内容提要、基本术语解释、常见问题解答等给出系统性的说明和描述，并在此基础上，提供了大量的单项选择练习题及其答案、分析应用题及其分析解答。

本书提供了系统性的教学指导和丰富的习题及其解答，可以作为高等学校计算机专业本科或高职高专学生计算机组成原理与系统结构课程的教学辅助教材，也可以作为有关专业研究生或计算机技术人员的参考书。

图书在版编目（CIP）数据

计算机组成与系统结构习题解答与教学指导/袁春风编著. —北京：清华大学出版社，2011.5
（21世纪大学本科计算机专业系列教材）
ISBN 978-7-302-24627-5

Ⅰ. ①计…　Ⅱ. ①袁…　Ⅲ. ①计算机组成原理－高等学校－教学参考资料 ②计算机体系结构－高等学校－教学参考资料　Ⅳ. ①TP30

中国版本图书馆CIP数据核字(2011)第041001号

责任编辑：张瑞庆　薛　阳
责任校对：梁　毅
责任印制：

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×260　**印　张**：17.5　**字　数**：436千字
版　次：2011年5月第1版　**印　次**：2011年5月第1次印刷
印　数：1～0000
定　价：0.00元

产品编号：-

前 言

FOREWORD

“计算机组成原理”和“计算机系统结构”是计算机学科各专业的核心基础课。有些高校将两门课程分开来上，也有些高校针对单处理器计算机系统开设了一门“计算机组成与系统结构”课程，而针对多核处理器、多处理器计算机、并行计算机等高级体系结构的内容专门开设一门“高级体系结构”课程。不管如何设置课程，计算机组成与计算机系统结构涉及的基本内容都是一样的，这些内容位于软、硬件结合处，不仅涉及到计算机硬件设计、计算机指令系统设计，还涉及到操作系统、编译器和高级语言程序设计等部分软件设计技术，因此，相关内容是整个计算机系统中最核心的部分。

2010 年 4 月由清华大学出版社出版的由作者编写的主教材《计算机组成与系统结构》主要针对单处理器计算机系统的组成与系统结构涉及的相关内容进行介绍。为了让学生更好地理解基本概念和基本原理，在主教材编写时，采用了“基本原理和实现细节相结合，历史发展过程和现实实际情况相结合”的方针，力求使学生能够全面、系统、准确、牢固地掌握相关知识。因而，使得主教材的内容涵盖面广、细节内容较多、篇幅较大，给教材的使用带来一些困难。

为了更好地协助主讲老师用好主教材，作者编写了本辅助教材，对主教材中每一章的内容进行了概括总结，给出了以下 6 个方面的教学辅助内容。

(1) 教学目标和内容安排：给出相应章节的教学总体目标和基本教学要求，并比较详细地说明课堂教学内容和学生课后阅读内容的安排，以及每章的主要教学思路或教学方法。

(2) 主要内容提要：对主教材中相应章节内容进行浓缩，形成主干知识框架结构，便于学生将全书内容串接起来，形成本课程的知识框架体系。

(3) 基本术语解释：给出相关章节所涉及的基本术语的解释说明，并给出名词术语的中英文对照。

(4) 常见问题解答：提供了大量的常见问题，并给出对每个问题的解释说明。这些常见问题是作者在长期的教学过程中发现的普遍存在于学生中的共性问题。

(5) 单项选择题：提供了相应章节内容的单项选择练习题及其参考答案。

(6) 分析应用题：提供了相应章节内容的分析应用题及其分析解答。

单项选择题和分析应用题两个方面的教学辅助内容，主要是为了巩固学生所学的基本原理而设置的。通过对一些具体问题的分析，能够提高学生对基本原理的认识。

本书作为主教材的教学辅助资料，可以与主教材配套使用。同时，本书相对独立、自成体系，因此也可单独使用。既可作为“计算机组成原理”或“计算机组成与系统结构”课程的

教师参考书，也可作为学生学习“计算机组成原理”或“计算机组成与系统结构”课程时的学习参考书。

在本书的编写过程中，得到了杨若瑜副教授的大力协助，从书稿的篇章结构到内容各方面都提出了许多宝贵的意见，并对全书内容进行了全面细致的核查和校对。此外，黄宜华教授、武港山教授等也对本书提出了许多宝贵的意见。杨晓亮、肖韬、翁基伟、刘长辉、宗恒、莫志刚、叶俊杰等研究生对相关章节的内容和习题分别进行了校对和试做，并提出了许多宝贵的意见和修改建议。在此对以上各位老师和研究生一并表示衷心的感谢。

由于计算机组成与系统结构相关的基础理论和技术在不断发展，新的思想、概念、技术和方法不断涌现，加之作者水平有限，在编写中难免存在不当或遗漏之处，恳请同行专家和广大读者对本书的不足之处给予指正，以便在后续的版本中予以改进。在主教材或本教材使用过程中若有任何问题，都可与作者直接联系，联系方式：cfyuan@nju. edu. cn。

作者于南京

2011 年元月

目 录

CONTENTS

第 1 章 计算机系统概论

1.1 教学目标和内容安排

主要教学目标：

概要了解整个计算机系统全貌以及本课程内容在计算机系统中的位置，并使学生掌握如何简单评价计算机系统的性能。

基本学习要求：

(1) 简单了解计算机的发展历程、计算机分代标志、摩尔定律的内容。

(2) 了解计算机系统中硬件和软件的基本概念及其相互关系。

(3) 了解冯·诺依曼结构计算机的特点，以及计算机硬件的基本组成和各部件功能。

(4) 了解计算机软件的分类，以及各类系统软件和应用软件的功能。

(5) 了解程序开发和执行过程，理解各种语言处理程序（翻译程序、编译程序、汇编程序）的概念。

(6) 理解计算机系统的层次化结构。

(7) 了解各类计算机用户在计算机系统中所处位置，以及本课程在计算机系统中所处位置。

(8) 了解如何对计算机的性能进行测量和评价。

(9) 了解有哪些因素会影响计算机的性能。

本章涉及的内容是计算机学科最基本的概念和知识，虽然没有特别难懂的部分，但是，对于低年级学生来说，有些概念还是比较抽象和难以理解的，需要在对后面章节的不断学习过程中，去深化对它们的理解并熟练运用。遇到这些内容时，可以告诉学生相关内容在后面具体哪个章节中会详细介绍。

本章有关计算机发展历程的部分内容中，出现了许多学生难以理解的专业术语，因此，这部分内容可以跳过不讲，但是，对于冯·诺依曼计算机结构的特点、“存储程序”工作方式、“兼容性”概念、摩尔定律等内容，要求学生能够掌握和理解。

计算机层次化概念和计算机系统组成的内容是相互联系的。不同计算机用户眼中的计算机系统是不一样的，可以从最终用户感觉到的计算机硬件和软件的形态开始，逐步深入到系统管理员、应用程序员、系统程序员，以及系统架构师眼中的硬件和软件形态。这两部分内容对学生建立整个计算机系统全貌以及了解本课程在计算机系统中的位置是非常重

要的。

为了让学生能够深入理解高级语言源程序和机器语言目标代码之间的关系，建议让学生做一些编程实验，例如，可以让学生对主教材[①] 1.3.5 节给出的 hello.c 源程序进行编译、链接，最终生成可执行目标代码，并要求学生比较源程序和目标代码，找出对应关系，并找到函数 printf ()对应的机器代码段等。

1.2 主要内容提要

1. 冯·诺依曼计算机结构的主要特点

冯·诺依曼计算机结构的主要特点包括：①计算机由运算器、控制器、存储器、输入设备和输出设备 5 大部分组成。②指令和数据用二进制表示，两者形式上没有差别。③指令和数据存放在存储器中，按地址访问。④指令由操作码和地址码组成，操作码指定操作性质，地址码指定操作数地址。⑤采用"存储程序"方式进行工作。

2. 计算机硬件的基本组成和功能

运算器用来进行各种算术逻辑运算；控制器用来对指令译码并送出操作控制信号；存储器用来存放指令和数据；输入和输出设备用来实现计算机和用户之间的信息交换。

3. 计算机系统的层次结构

计算机系统分软件和硬件两部分，软件和硬件的界面是指令集体系结构(ISA)。计算机系统从高到低粗分为应用软件、系统软件和硬件 3 个层次；不同计算机用户工作在不同的层次，从高到低细分为应用程序级(最终用户)、高级语言虚拟机级(高级语言程序员或应用程序员)、汇编语言虚拟机级(汇编语言程序员)、操作系统虚拟机级(系统管理员)、机器语言机器级(机器语言程序员)。

4. 硬件和软件的相互关系

计算机硬件和软件两者相辅相成，缺一不可。两者都用来实现逻辑功能，同一功能可用硬件实现，也可用软件实现。

5. 程序开发和执行过程

首先用某种语言(高级语言或低级语言)编制源程序，然后用语言处理程序(编译程序或汇编程序)将源程序翻译成机器语言目标程序。通过某种方式启动目标程序(可执行代码)执行时，操作系统将指令和数据装入内存，然后从第一条指令开始执行。每条指令的执行过程为取指令、指令译码、取操作数、运算、送结果、PC 指向下一条指令。可执行程序由若干指令组成，CPU 周而复始地执行一条一条指令，直到程序所含指令全部执行完。

6. 基本性能指标和性能参数

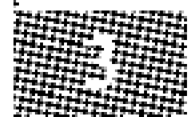
计算机系统的基本性能指标包括响应时间和吞吐率。处理器的基本性能参数包括时钟周期(或主频)、CPI、MIPS、MFLOPS(或 GFLOPS、TFLOPS、PFLOPS 等)。

7. 性能的测量

一般把程序的响应时间划分成 CPU 时间和等待时间，CPU 时间又分成用户 CPU 时间和操作系统 CPU 时间。因为操作系统对自身所用的时间进行测量时，不十分准确，所以，

---

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)



对 CPU 性能的测量一般是通过测量程序运行时的用户 CPU 时间来进行的。

8. 各种性能指标之间的关系

时钟周期和时钟频率互为倒数。CPU 执行时间可用“CPU 时钟周期数×时钟周期”来计算。CPU 时钟周期数可用“程序所含指令条数×程序的 CPI”来计算。

9. 性能评价程序

可采用一组性能评价程序(即基准程序)对机器性能进行评测。有些机器制造商会针对基准程序中频繁出现的语句采用专门的编译器进行优化,使基准程序的运行效率大幅提高,因此有时用基准程序得到的评测结果不能说明问题。

1.3 基本术语解释

系列机(Family Machine)

系列机是指一个厂家生产的具有相同系统结构、不同组成和实现的一系列不同型号的机器。它应在指令系统、数据格式、字符编码、中断系统、控制方式、输入/输出控制方式等方面保持统一,从而保证软件的兼容性。

兼容性(Compatibility)

兼容是一个广泛的概念,主要表示一种“互换”特性,包括软件兼容、硬件兼容等。

软件兼容(Software Compatibility)

指在某档机型上开发的软件可以不加修改地在另外的机型上正确运行。一般在同一系列机型内的软件是兼容的,有向上兼容和向下兼容两种形式。向上兼容是指高档机型上的程序能在低档机型上运行,向下兼容是指低档机型上的程序能在高档机型上运行。一般系列机满足向下兼容性,因为系列机中高档机的指令系统包含了低档机中的所有指令。

硬件兼容(Hardware Compatibility)

也就是设备或部件兼容,是指设备或部件可以不加改动地用于多种计算机。这要求设备或部件符合某种标准化设计。

小规模集成电路(Small Scale Integrated Circuits,SSI)

集成度小于 100 的集成电路。

中规模集成电路(Medium Scale Integrated Circuits,MSI)

集成度在 100 到 1000 的集成电路。

大规模集成电路(Large Scale Integrated Circuits,LSI)

集成度在 1000 到 10 万的集成电路。

超大规模集成电路(Very Large Scale Integrated Circuits,VLSI)

集成度在 10 万到 1000 万的集成电路。

极大规模集成电路(Ultra Large Scale Integrated Circuits,ULSI)

集成度在 1000 万以上的集成电路。

摩尔定律(Moore's Law)

1965 年摩尔预测:“以后每年将缩小硅片中形成晶体管电路的细线尺寸的 10%,芯片制造商能够每 3 年发布新一代的芯片,其晶体管数为上一代的 4 倍”。后来摩尔定律被表述成:“由于集成电路技术的不断改进,每 18～24 个月,集成电路芯片上集成的晶体管数将翻

一番，速度将提高一倍，而价格将降低一半”。

通用电子计算机(General-purpose Electronic Computer)

通用电子计算机是和专用电子计算机对应的，专用机只能专门用于某种应用，而通用电子计算机，从定义上来说，它可以解决任何问题，只要这个问题可以用程序来表示。通用电子计算机也被称为完备的图灵机。

算术逻辑单元(Arithmetic Logic Unit, ALU)

对数据进行算术运算和逻辑运算处理的部件。

数据通路(Datapath)

数据通路是指指令在执行过程中数据所经过的部件以及部件之间的连接线路，主要由 ALU 和一组寄存器、存储器、总线等组成。国内许多教科书中提到的运算器即运算数据通路。

控制器(Control Unit)

也称为控制单元或控制部件。其作用是对指令进行译码，将译码结果和状态/标志信号、时序信号等进行组合，产生各种操作控制信号。这些操作控制信号被送到 CPU 内部或通过总线送到主存或 I/O 模块。送到 CPU 内部的控制信号用于控制 CPU 内部数据通路的执行，送到主存或 I/O 模块的信号控制 CPU 和主存或 CPU 和 I/O 模块之间的信息交换。控制单元是整个 CPU 的指挥控制中心，通过规定各部件在何时做什么动作来控制数据的流动，完成指令的执行。

中央处理器(Central Processing Unit, CPU)

中央处理器是计算机中最重要的部分之一，主要由运算器和控制器组成。其内部结构归纳起来可以分为控制部件、运算部件和存储部件 3 大部分，它们相互协调，共同完成对指令的执行。

存储器(Memory, Storage)

存储器用于存储程序和数据，分为内存储器(Memory)和外存储器(Storage)。内存存取速度快、容量小、价格贵；外存容量大、价格低，但存取速度慢。

内存(Memory)

从字面上来说，内存是内部存储器，应该包括主存(Main Memory, MM)和高速缓存(cache)，但是，因为早期计算机中没有高速缓存，因而传统意义上内存就是主存，所以，目前也并不区分内存和主存。内存位于 CPU 之外，用来存放已被启动执行的程序及所用数据，包括 ROM 芯片和 RAM 芯片组成的相应 ROM 存储区和 RAM 存储区两部分。

外存(Storage)

外存储器主要有磁盘存储器、磁带存储器和光盘存储器等。磁盘是最常用的外存储器，通常它分为软盘和硬盘两类。容量极大、价格便宜的磁带机和光盘组等称为海量存储器，常用作数据备份，也称为辅存(Accessorial Memory, AM)或二级存储器(Secondary Memory)。

系统软件(System Software)

系统软件是介于计算机硬件与应用程序之间的各种软件，它与具体应用关系不大。系统软件包括操作系统(如 Windows)、语言处理系统(如 C 语言编译器)、数据库管理系统(如 Oracle)和各类实用程序(如磁盘碎片整理程序、备份程序)。

应用软件(Application Software)

应用软件是指为针对使用者的某种应用目的所编写的软件,例如,办公自动化软件,互联网应用软件,多媒体处理软件,股票分析软件,游戏软件,管理信息系统等。

操作系统(Operating System)

操作系统简称OS,是计算机系统中负责支撑应用程序运行环境以及用户操作环境的系统软件,其目的是使计算机系统所有资源最大限度地发挥作用,并为用户提供方便、有效、友善的服务界面。操作系统是一个庞大的管理控制程序,大致包括5个方面的管理功能:进程与处理机管理、作业管理、存储管理、设备管理和文件管理。目前比较流行的操作系统主要有两个家族:类Unix家族和微软的Windows家族。

最终用户(End User)

使用应用软件完成特定任务的计算机用户称为最终用户。大多数计算机使用者都属于最终用户。例如,使用炒股软件的股民、玩计算机游戏的人、进行会计电算化处理的财会人员等。

系统管理员(System Administrator)

利用操作系统提供的功能对系统进行配置、管理和维护以建立高效合理的系统环境供计算机用户使用的操作人员。其职责主要包括安装、配置和维护系统的硬件和软件,建立和管理用户账户,升级软件,备份和恢复业务系统及数据等。

应用程序员(Application Programmer)

使用高级编程语言编制应用软件的程序员。

系统程序员(System Programmer)

设计和开发系统软件的程序员,如开发操作系统、编译器、数据库管理系统等系统软件的程序员。

高级语言(High-level Programming Language)

也称高级编程语言或算法语言。它是面向问题和算法的描述语言。用这种语言编写程序时,程序员不必了解实际机器的结构和指令系统等细节,而通过一种比较自然的、直接的方式来描述问题和算法。

汇编语言(Assembly Language)

汇编语言是一种面向实际机器结构的低级语言,是机器语言的符号表示,与机器语言一一对应。因此,汇编语言程序员必须对机器的结构和指令系统等细节非常清楚。

机器语言(Machine Language)

机器语言是指直接用二进制代码(指令)表示的语言。用户必须用二进制代码来编写机器语言程序。因此,机器语言程序员必须对机器的结构和指令系统等细节非常清楚。

指令集(Instruction Set)

一台计算机能够执行的所有机器指令的集合。按功能指令可以分为运算指令、移位指令、传送指令、串指令、程序控制指令等类型。

指令集体系结构(Instruction Set Architecture,ISA)

是计算机硬件与系统软件之间的接口,指机器语言程序员或操作系统、编译器、解释器设计人员所看到的计算机功能特性和概念性结构。其核心部分是指令系统,同时还包含数据类型和数据格式定义、寄存器设计、I/O空间的编址和数据传输方式、中断结构、计算机状

态的定义和切换、存储保护等。ISA设计的好坏直接决定了计算机的性能和成本。

透明性(Transparency)

由于计算机系统采用了层次化结构进行设计和组织，因而面向不同的硬件或软件层面进行工作的人员或用户所“看到”的计算机是不一样的。也就是说，计算机组织方式或系统结构中的一部分对某些用户而言是“看不到”的或称为“透明”的。例如，对于高级语言程序员来说，指令格式、数据格式、机器结构、指令和数据的存取方式等，都是透明的；而对于机器语言程序员和汇编语言程序员来说，指令格式、机器结构、数据格式等则不是透明的。

源程序(Source Program)

编译程序、解释程序和汇编程序统称为语言处理程序。各种语言处理程序处理的对象称为源程序，用高级(算法)语言或汇编语言编写。如C语言源程序、Java语言源程序、汇编语言源程序等。

目标程序(Object Program)

编译程序和汇编程序对源程序进行翻译得到的结果程序称为目标程序或目标代码(Object Code)。

编译程序(Compiler)

也称编译器，用来将高级语言源程序翻译成汇编语言或机器语言目标代码的程序。

解释程序(Interpreter)

解释程序将源程序的一条语句翻译成对应的机器语言目标代码，并立即执行，然后翻译下一条源程序语句并执行，直到所有源程序中的语句全部被翻译并执行完。因此，解释程序并不输出目标程序，而是直接输出源程序的执行结果。

汇编程序(Assembler)

汇编程序也是一种语言翻译程序，它把汇编语言写的源程序翻译为机器语言目标程序。汇编程序和汇编语言是两个不同的概念，不能混为一谈。

响应时间(Response Time)

也称执行时间(Execution Time)或延迟时间(Latency)，是指从作业提交开始到作业完成的时间。一般一个程序的响应时间除了CPU执行程序包含的指令执行时间外，还包括等待I/O的时间，系统运行其他用户程序所用的时间，以及操作系统运行的时间等。

吞吐率(Throughput)

在有些场合下，吞吐率也称带宽(Bandwidth)，是指在一定的时间内所完成的工作量。

CPU执行时间(CPU Execution Time)

也称CPU时间，指在程序运行过程中，CPU真正用于程序执行的时间。它不包括因为等待I/O操作完成所需要的时间，也不包括CPU执行其他程序所需要的时间。对用户来说，能直接感觉到的只能是响应时间，而不可能是CPU执行时间。它被进一步分为两部分：一部分是用来运行用户程序代码的时间，称为用户CPU时间(User CPU Time)；另一部分是为了执行用户程序而运行一些操作系统代码所花费的时间，称为系统CPU时间(System CPU Time)。

系统性能(System Performance)

通常用在没有其他任何负载的情况下程序的响应时间来表示系统性能。

CPU性能(CPU Performance)

通常用用户 CPU 时间来表示 CPU 性能。

时钟周期(Clock Cycle,Tick,Clock Tick,Clock)

所有计算机执行指令的过程都是分成若干步骤和相应的动作来完成的,每一步动作都可能有相应的控制信号进行控制,这些控制信号何时发出、作用时间多长,都要有相应的定时信号进行同步。因此,CPU 必须能够产生用于同步的时钟定时信号,也就是 CPU 的主脉冲信号,其宽度称为时钟周期。

时钟频率(Clock Rate,主频)

CPU 的主频就是 CPU 中的主脉冲时钟信号的频率,是 CPU 时钟周期的倒数。

CPI(Cycle Per Instruction)

衡量 CPU 性能的一种计量单位。表示执行一条指令所需要的平均时钟周期个数。

基准测试程序(Benchmark)

是专门用来进行性能评价的一组程序,这些程序能够很好地反映机器在运行实际负载时的性能。可以在不同机器上运行相同的基准测试程序来比较不同机器的运行时间,从而比较其性能。

SPEC 基准测试程序集(SPEC Benchmark)

系统性能评价标准 SPEC(System Performance Evaluation Cooperative)是一个应用最广泛,也是最全面地对 CPU 性能进行评测的基准程序集。分整数程序集 SPECint 和浮点程序集 SPECfp。

SPEC 比值(SPEC Ratio)

将测试程序在 Sun SPARCstation 上运行时的执行时间除以该程序在测试机器上的执行时间所得到的比值。比值越大,机器的性能越好。

阿姆代尔定律(Amdahl's Law)

主要含义是指系统优化某部件所获得的系统性能的改善程度,取决于该部件被使用的频率,或所占总执行时间的比例。该定律很好地刻画了改善"系统瓶颈"性能的重要性。

MIPS (Million Instructions Per Second)

用来衡量单位时间内执行指令的条数,具体是指每秒钟执行多少百万条指令。

加法指令执行速度(Add Instruction Execution Speed)

最早用来衡量计算机性能的指标是完成单个运算(如加法运算)指令所需要的时间。当时大多数指令的执行时间是相同的。加法指令能反映乘、除等运算,而其他指令的时间也大体与加法指令相当。故加法指令的速度有一定的代表性。计量单位为 KIPS(每秒千条指令)和 MIPS (每秒百万条指令)。

平均指令执行时间(Average Instruction Execution Time)

也称等效指令速度法或 Gibson 混合法。随着计算机的发展,不同指令所需要的执行时间差别越来越大,人们就根据等效指令速度法通过统计各类指令在程序中所占比例进行折算。设某类指令 i 在程序中所占比例为 w_i,执行时间为 t_i,则等效指令的执行时间为 $T=w_1\times t_1+w_2\times t_2+\cdots+w_n\times t_n$($n$ 为指令种类数)。如果指令执行时间用时钟周期数来衡量,平均指令执行时间就是平均 CPI。对平均指令执行时间求倒数能够得到平均 MIPS。

峰值 MIPS(Peak MIPS)

选取一组指令组合,使得平均 CPI 最小,由此得到的 MIPS 就是峰值 MIPS。有些制造

商经常将峰值 MIPS 直接当作 MIPS，因此，实际上的性能要比标称的性能差。

相对 MIPS(Relative MIPS)

根据一种公认的参考机型来定义相应的 MIPS 值，其值的含义是相对于参考机型 MIPS 的多少倍。

MFLOPS(Million Floating-point Operations Per Second)

是计算机浮点运算速度的一种计量单位。表示每秒钟执行多少百万次浮点操作。它是基于所完成的单精度浮点数的操作次数而不是指令数来衡量的。

1.4 常见问题解答

1. 计算机系统就是硬件系统吗?

答：说计算机系统就是硬件系统是不完整的。一个完整的计算机系统应该包括硬件系统和软件系统两部分。硬件系统包括运算器、控制器、存储器、输入设备和输出设备五大基本部件。软件系统分为系统软件和应用软件两大类。系统软件包括操作系统、计算机语言处理程序(各种程序翻译软件，包括编译程序、解释程序、汇编程序)、服务性程序、数据库管理系统和网络软件等；应用软件包括各种特定领域的处理程序。计算机系统中的硬件和软件是相辅相成的，缺一不可。软件是计算机系统的灵魂，没有软件的硬件不能被用户使用。

2. 同一个功能可以由软件完成也可以由硬件完成吗?

答：软件和硬件是两种完全不同的形态，硬件是实体，是物质基础；软件是一种信息，看不见，摸不到。但是它们都可以用来实现逻辑功能，所以在逻辑功能上，软件和硬件是等价的。因此，在计算机系统中，许多功能既可以直接由硬件实现，也可以在硬件的配合下由软件来实现。例如：乘法运算既可以用专门的乘法器硬件实现，即机器提供专门的一条乘法指令；也可以用乘法子程序来实现，即不提供乘法指令，而由加法指令和移位指令等组成的一个指令序列来完成乘法运算。

3. 解释程序和编译程序有什么差别?

答：编译程序和解释程序是两种不同的翻译程序。不同在于编译程序将高级语言源程序全部翻译成目标程序，每次执行程序时，只要执行目标程序，因此，只要源程序不变，就无须重新翻译；解释程序是将源程序的一条语句，翻译成对应的机器目标代码并立即执行，然后翻译下一条语句并执行，直到所有源程序中的语句全部被翻译并执行完。所以解释程序的执行过程是翻译一句，执行一句。解释的结果是源程序执行的结果，而不会生成目标程序。

4. 要计算机做的任何工作都要先编写成程序才能完成吗?

答：是的。要计算机完成的任何事情，都必须先编制程序，程序是由指令构成的。不管是用哪种语言编写的程序，最终都要翻译成机器语言程序才能让机器理解。机器语言程序是由一条一条指令组成的程序。CPU 的主要功能就是周而复始地执行指令，因此，要计算机完成的所有功能都是通过执行一条一条指令来实现的，也就是由一个程序来完成的。有时说某个特定的功能是由硬件实现的，并不是说不要编写程序，如乘法功能可由乘法器这个硬件实现，但要启动这个硬件(乘法器)工作，必须先执行程序中的乘法指令。

5. 指令和数据在形式上没有差别，且都存于存储器中，计算机如何区分它们呢？

答：指令和数据在计算机内部都是用二进制表示的，因而都是0、1序列，在形式上没有差别。在指令和数据取入到CPU之前，它们都存放在存储器中，CPU必须能够区分读出的是指令还是数据。如果是指令，CPU会把指令的操作码送到指令译码器进行译码，而把指令的地址码送到相应的地方进行处理；如果是数据，则送到寄存器或运算器。那么，CPU如何识别读出的是指令还是数据呢？实际上，CPU并不是把信息从主存读出后，靠某种判断方法来识别信息是数据还是指令的，而是在读出之前就知道将要读的信息是数据还是指令了。执行指令的过程分为取指令、指令译码、取操作数、运算、送结果等。所以，在取指令阶段，总是将程序计数器PC的内容作为地址去取指令，所以取来的一定是指令；取操作数阶段取来的一定是数据。

6. 衡量计算机系统性能的主要指标是什么？

答：计算机性能的好坏主要体现在速度的快慢，而衡量速度快慢主要有两个指标：响应时间和吞吐率。响应时间是指从作业提交开始到作业完成所花的时间。一般一个程序的响应时间除了CPU执行该程序包含的指令所用的时间外，还包括等待输入输出操作所需时间和操作系统运行该程序所花的时间开销等。吞吐率是指单位时间内完成的工作量。

7. CPU的时钟频率越高，机器的速度就越快，对吗？

答：在其他因素不变的情况下，CPU的时钟频率越高，机器的速度肯定越快。但是，程序执行的速度除了与CPU的速度有关外，还与存储器和I/O模块的存取速度、总线的传输速度、cache的设计策略等都有很大关系。因此，机器的速度不是只由CPU的时钟频率决定的。

8. CPI的含义是什么？执行时间（响应时间）与CPI是什么关系？

答：CPI是指每条指令执行所用的时钟周期数。通常，一条特定指令的CPI是一个确定的值，而某个程序的CPI则是一个平均值。一个程序的执行时间取决于该程序所包含的指令数、CPI和时钟周期。在指令条数和时钟周期一定的情况下，CPI越大，执行时间越长。

9. 用户真正感觉到的程序执行时间是否就是执行一个程序中所有指令所用的时间？

答：不是。执行一个程序中所有指令所用的时间实际上比用户真正感觉到的程序执行时间要短。因为在一个程序执行过程中，可能还会执行操作系统代码或其他用户程序，并且，有时还可能等待I/O操作。例如，在UNIX操作系统中，假定用time命令测试某程序执行时间的结果为"80.3u 10.1s 2：02 74%"，则说明该程序所有指令的执行时间（即用户CPU时间）只有80.3秒，执行操作系统代码所用时间（即系统CPU时间）是10.1秒，用户感觉到的总响应时间为$2\times60+2=122$秒，其中的$(80.3+10.1)/122=74\%$是CPU时间，剩下26%的时间（30多秒钟）用来等待I/O操作或运行其他用户程序。

10. 计算机的MIPS数越大，说明性能越好，对吗？

答：不对。MIPS数反映的是机器执行定点指令的速度。但是，不同机器的指令集不同，指令的功能也不同，也许一个机器上一条指令的功能，在另外一个机器上要用多条指令来完成，这样，同样的指令条数所完成的功能可能完全不同。因此用MIPS数来对不同的机器进行性能比较是不太客观的。

11. 基准测试程序执行得越快，说明机器的性能越好，对吗？

答：一般情况下，基准测试程序能够反映机器性能的好坏。但是，如果制造商针对基准测试程序中频繁出现的语句采用专门的编译器，使基准程序的运行效率大幅提高，那么基准评测程序测试的结果就不能说明问题。

1.5 单项选择题

1. 以下有关对摩尔定律的描述中，错误的是(　　)。
   A. 每18个月，集成电路芯片上集成的晶体管数将翻一番
   B. 每18个月，集成电路芯片的速度将提高一倍
   C. 每18个月，集成电路芯片的价格将降低一半
   D. 集成电路技术一直会遵循摩尔定律发展下去
2. 从计算机的主要元器件来看，计算机发展所经历的过程为(　　)。
   A. 晶体管、电子管、SSI、MSI、LSI、ULSI、VLSI
   B. 电子管、晶体管、SSI、MSI、LSI、VLSI、ULSI
   C. 电子管、晶体管、LSI、MSI、SSI、VLSI、ULSI
   D. 晶体管、电子管、MSI、SSI、LSI、ULSI、VLSI
3. 一个完整的计算机系统包括硬件和软件。软件又分为(　　)。
   A. 操作系统和语言处理程序
   B. 系统软件和应用软件
   C. 操作系统和高级语言
   D. 低级语言程序和高级语言程序
4. 以下给出的软件中，属于应用软件的是(　　)。
   A. 汇编程序　　　　B. 编译程序
   C. 操作系统　　　　D. 文字处理程序
5. 以下给出的软件中，属于系统软件的是(　　)。
   A. Windows XP　　　　B. MS Word
   C. 金山词霸　　　　D. RealPlayer
6. 下面有关指令集体系结构的说法中，错误的是(　　)。
   A. 指令集体系结构位于计算机软件和硬件的交界面上
   B. 指令集体系结构是指低级语言程序员所看到的概念结构和功能特性
   C. 程序员可见寄存器的长度、功能与编号不属于指令集体系结构的内容
   D. 指令集体系结构的英文缩写是ISA
7. 计算机系统采用层次化结构，从最上面的应用层到最下面的硬件层，其层次化构成为(　　)。
   A. 高级语言虚拟机—操作系统虚拟机—汇编语言虚拟机—机器语言机器
   B. 高级语言虚拟机—汇编语言虚拟机—机器语言机器—操作系统虚拟机
   C. 高级语言虚拟机—汇编语言虚拟机—操作系统虚拟机—机器语言机器
   D. 操作系统虚拟机—高级语言虚拟机—汇编语言虚拟机—机器语言机器

8. 以下有关程序编写和执行方面的叙述中，错误的是（　　）。
   A. 可用高级语言和低级语言编写出功能等价的程序
   B. 高级语言源程序和汇编语言源程序都不能在机器上直接执行
   C. 编译程序员必须了解机器结构和指令系统
   D. 汇编语言是一种与机器结构无关的编程语言
9. 冯·诺依曼计算机中，CPU 区分从存储器取出的是指令还是数据的依据是（　　）。
   A. 指令译码结果的不同
   B. 访问指令和访问数据时寻址方式不同
   C. 访问指令和访问数据时所处的指令执行阶段不同
   D. 指令和数据所在的存储单元地址范围不同
10. 以下有关冯·诺依曼结构计算机指令和数据表示的叙述中，正确的是（　　）。
   A. 指令和数据可以从形式上加以区分
   B. 指令以二进制形式存放，数据以十进制形式存放
   C. 指令和数据都以二进制形式存放
   D. 指令和数据都以十进制形式存放
11. 以下是有关计算机中指令和数据存放位置的叙述，其中正确的是（　　）。
   A. 指令存放在内存，数据存放在外存
   B. 指令和数据任何时候都存放在内存
   C. 指令和数据任何时候都存放在外存
   D. 程序被启动后，其指令和数据才被装入内存
12. 冯·诺依曼计算机工作方式的基本特点是（　　）。
   A. 程序从键盘输入的同时被计算机执行
   B. 程序直接从磁盘被读到 CPU 执行
   C. 程序中的指令按地址被访问并自动按序执行
   D. 程序被自动执行而数据通过手工输入
13. 以下是有关冯·诺依曼计算机结构的叙述，其中错误的是（　　）。
   A. 计算机由运算器、控制器、存储器和输入/输出设备组成
   B. 程序由指令和数据构成，存放在存储器中
   C. 指令由操作码和地址码两部分组成
   D. 指令按地址访问，所有数据在指令中直接给出
14. 以下有关计算机各部件功能的叙述中，错误的是（　　）。
   A. 运算器用来完成算术运算
   B. 存储器用来存放指令和数据
   C. 控制器通过执行指令来控制整个机器的运行
   D. 输入/输出设备用来完成用户和计算机之间的信息交换
15. 以下给出了改善计算机性能的四种措施：
   ① 用更快的处理器来替换原来的慢速处理器
   ② 增加同类处理器个数，使得不同的处理器同时执行不同的程序
   ③ 优化编译生成的代码使得程序执行的总时钟周期数减少

④ 减少指令执行过程中访问内存的时间

对于某个特定的程序，以上措施中，能缩短其执行时间的措施是(　　)。

A. ①、②、③　　B. ①、②、④

C. ①、③、④　　D. 全部

16. 若某典型基准测试程序在机器 A 上运行时需要 20s，而在机器 B 上的运行时间是 16s，那么，相对来说，下面给出的结论中，正确的是(　　)。

A. 所有程序在机器 A 上都比在机器 B 上运行速度慢

B. 机器 B 的速度是机器 A 的 1.25 倍

C. 机器 A 的速度是机器 B 的 1.25 倍

D. 机器 A 比机器 B 慢 1.25 倍

17. 已知计算机 A 的时钟频率为 800MHz，假定某程序在计算机 A 上运行需要 12 秒钟。现在硬件设计人员想设计计算机 B，希望该程序在 B 上的运行时间能缩短为 8 秒钟，使用新技术后可使 B 的时钟频率大幅度提高，但在 B 上运行该程序所需要的时钟周期数为在 A 上的 1.5 倍。那么，机器 B 的时钟频率至少应为多少才能达到所希望的要求？(　　)

A. 800MHz　　B. 1.2GHz　　C. 1.5GHz　　D. 1.8GHz

18. 假设同一套指令集用不同的方法设计了两种计算机 A 和 B。机器 A 的时钟周期为 1.2ns，机器 B 的时钟周期为 2ns。某个程序在机器 A 上运行时的 CPI 为 2，在 B 上的 CPI 为 1。则对于该程序来说，机器 A 和机器 B 之间的速度关系为(　　)。

A. 机器 A 比机器 B 快 1.2 倍

B. 机器 B 比机器 A 快 1.2 倍

C. 机器 A 的速度是机器 B 的 1.2 倍

D. 机器 B 的速度是机器 A 的 1.2 倍

19. 假定编译器对高级语言的某条语句可以编译生成两种不同的指令序列，A、B 和 C 三类指令的 CPI 和执行两种不同序列所含的三类指令条数见表 1.1。

表 1.1　各类指令的 CPI 及执行的指令条数

指　令　类	CPI	序列一的指令条数	序列二的指令条数
A	1	2	4
B	2	1	1
C	3	2	1

则以下哪个结论是错误的？(　　)

A. 序列一比序列二少 1 条指令

B. 序列一比序列二的执行速度快

C. 序列一的总时钟周期数比序列二多 1 个

D. 序列一的 CPI 比序列二的 CPI 大

20. 假定用不同的编译器对同一个程序进行编译生成不同的目标代码指令序列，A、B 和 C 三类指令的 CPI 和执行两种不同指令序列所含的三类指令条数见表 1.2。

表 1.2 各类指令的 CPI 及执行的指令条数

指令类	CPI	序列一的指令条数	序列二的指令条数
A	1	5×10^9	10×10^9
B	2	1×10^9	1×10^9
C	3	1×10^9	1×10^9

两个指令序列都在时钟周期为 2ns 的机器上运行。根据计算得到其 MIPS 指标和执行速度两方面的结论为(　　)。

A. 序列一的 MIPS 数比序列二多 50,序列一的执行速度也比序列二快 10s

B. 序列二的 MIPS 数比序列一多 50,但序列一的执行速度比序列二快 10s

C. 序列一的 MIPS 数比序列一多 100,序列一的执行速度也比序列二快 20s

D. 序列二的 MIPS 数比序列一多 100,但序列一的执行速度比序列二快 20s

【参考答案】

1. D　2. B　3. B　4. D　5. A　6. C　7. C
8. D　9. C　10. C　11. D　12. C　13. D　14. A
15. D　16. B　17. D　18. D　19. B　20. B

1.6 分析应用题

1. 请说明以下措施对缩短程序的响应时间和提高系统的吞吐率各有什么影响?

(1) 使用更快的处理器。

(2) 增加处理器个数,使得不同的处理器同时处理不同的作业。

(3) 优化编译生成的代码使得程序执行的总时钟周期数减少。

(4) 在 CPU 和主存之间增加 cache,减少 CPU 访问指令和数据的时间。

【分析解答】

措施(1):因为总执行时间=总时钟周期数×时钟周期。更快的处理器意味着时钟频率加快了,即每个时钟周期更短,故总执行时间变短了。因此采用更快的处理器可缩短程序响应时间,从而使单位时间内完成的作业数增加,系统的吞吐率也提高了。

措施(2):处理器个数的增加使得同时可以有多个作业在不同的处理器上进行处理,因而,系统在单位时间内可以完成更多的作业,吞吐率会明显提高。但每个作业的处理过程还是一样的,所以程序的执行时间并不会缩短。但是,因为吞吐率提高了,所以作业在等待队列中的排队时间减少了,因而响应时间会在一定的程度上有相应的改善。

措施(3):优化编译使生成的代码总的执行时钟周期数减少,也就缩短了程序的执行时间,因而使得单位时间内完成的作业数增加,系统的吞吐率也提高了。

措施(4):在 CPU 和主存之间增加 cache,使得程序访问存储器的时间大大缩短,因而可以缩短程序的响应时间,从而也就使得单位时间内完成的作业数增加,并提高了系统的吞吐率。

综上所述,措施(1)、(3)和(4)是因为首先缩短了程序的执行时间而使得系统吞吐率提

高；而措施(2)是因为提高了系统的吞吐率，使得程序等待时间缩短而缩短了程序响应时间。程序响应时间和系统吞吐率之间通常是相辅相成的关系，但在某些特定情况下，它们有可能是对立关系。

2. 假定某程序P由一个100条指令构成的循环组成，该循环共执行50次，在某系统S中执行程序P花了20000个时钟周期，则系统S在执行程序P时的CPI是多少？

【分析解答】

程序P中100条指令的循环被执行了50次，所以在20000个时钟周期中共执行了5000条指令，所以，系统S在执行程序P时CPI＝20000/5000＝4。

3. 假定执行某种指令I需要20个时钟周期，该指令在程序中的出现频度为10%，其他所有指令的平均CPI为5，则CPU执行指令I所用时间占整个CPU时间的百分比是多少？如果通过对硬件进行改进，能使指令I的执行时间缩短为10个时钟周期，但同时会使CPU时钟周期延长10%，你认为是否应该采取这种改进措施？

【分析解答】

假定CPU执行的所有指令条数为N，时钟周期为T，则CPU执行总时间为(20×10%＋5×90%)×N×T，其中指令I所用时间为20×10%×N×T，因此，CPU执行指令I所用时间占整个CPU时间的百分比是20×10%×N×T/((20×10%＋5×90%)×N×T)＝30.77%。如果改进后将指令I的执行时间缩短为10个时钟周期，时钟周期的长度延长10%，则改进后CPU执行的总时间为(10×10%＋5×90%)×N×1.1×T＝6.05×N×T，改进前的总时间为(20×10%＋5×90%)×N×T＝6.5×N×T，因而，改进后的性能是改进前的6.5/6.05＝1.07倍。由此可见，对硬件的这种改进措施能够提高CPU的性能。

4. 若有两个基准测试程序P1和P2在机器M1和M2上运行，假定M1和M2的价格分别是5000元和8000元，表1.3给出了P1和P2在M1和M2上所用的时间和指令条数。

表1.3　P1和P2在M1和M2上所用的时间和指令条数

程序	M1		M2	
	指令条数	执行时间	指令条数	执行时间
P1	200×10^6	1000ms	150×10^6	500ms
P2	300×10^3	3ms	420×10^3	6ms

请回答下列问题：

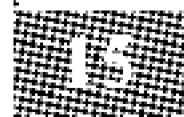
(1) 对于P1，哪台机器的速度快？快多少？对于P2呢？

(2) 在M1上执行P1和P2的速度分别是多少MIPS？在M2上的执行速度又各是多少？从执行速度来看，对于P2，哪台机器的速度快？快多少？

(3) 假定M1和M2的时钟频率各是800MHz和1.2GHz，则在M1和M2上执行P1时的平均时钟周期数CPI各是多少？

(4) 如果某个用户需要大量使用程序P1，并且该用户主要关心系统的响应时间而不是吞吐率，那么，该用户需要大批购进机器时，应该选择M1还是M2？为什么？(提示：从性价比上考虑)

(5) 如果另一个用户也需要购进大批机器，但该用户使用P1和P2一样多，主要关心的



也是响应时间,那么,应该选择 M1 还是 M2? 为什么?

【分析解答】

(1) 对于程序 P1,M1 所花的执行时间是 M2 的 2 倍,故 M2 比 M1 快 1 倍;对于程序 P2,M2 所花的执行时间是 M1 的 2 倍,故 M1 比 M2 快 1 倍。

(2) M1 中 P1 速度为 200M/1s=200MIPS,P2 速度为 300k/0.003s=100MIPS;M2 中 P1 速度为 150M/0.5s=300MIPS,P2 速度为 420k/0.006s=70MIPS。从执行速度来看,对于 P2,因为 100/70=1.43,所以 M1 比 M2 快 0.43 倍。

(3) 在 M1 上执行 P1 时的平均时钟周期数 CPI 为 1s×800MHz/200M=4;在 M2 上执行 P1 时的平均时钟周期数 CPI 为 0.5s×1.2GHz/150M=4。

(4) 因为该用户主要关心系统的响应时间,所以性价比中的性能主要考虑执行时间。若性能为执行时间的倒数,则性价比 R=1/(执行时间×价格)。R 越大说明性价比越高,也即"执行时间×价格"的值越小,则性价比越高。

对于程序 P1,M1 的性价比 $R1$=1/(1s×5000),M2 的性价比 $R2$=1/(0.5s×8000),根据计算,可知 $R2>R1$,故 M2 的性价比高,应选择购买 M2。

(5) 因为 P1 和 P2 需要同等考虑,所以需要考虑综合性能。有多种计算综合性能的方法,如执行时间总和、执行时间算术平均值、执行时间几何平均值等。

若用执行时间总和,则 M1 的性价比为 $R1$=1/(1003ms×5000),M2 的性价比为 $R2$=1/(506ms×8000),显然 $R2>R1$,故 M2 的性价比高,应选择 M2。

若用算术平均值,则 M1、M2 上执行时间的算术平均值分别为 501.5ms 和 253ms。因此,M1 的性价比为 $R1$=1/(501.5ms×5000),M2 的性价比为 $R2$=1/(253ms×8000),显然 $R2>R1$,故 M2 的性价比高,应选择 M2。

若用几何平均值,则 M1、M2 上执行时间的几何平均值都一样,约为 54.7。因此,M1 的性价比为 $R1$=1/(54.7×5000),M2 的性价比为 $R2$=1/(54.7×8000),显然 $R1>R2$,故 M1 的性价比高,应选择 M1。

由此可见,用不同的综合性能计算方法得到的结论可能不同。

5. 假定 M1 和 M2 是以不同方式实现同一个指令集的两种机器,M1 的时钟频率为 800MHz,M2 的时钟频率为 400MHz。在该指令集中一共有 A、B 和 C 三类指令。有 3 种不同的编译器,其中 C1 和 C2 分别是 M1 和 M2 的生产厂商提供的,C3 是第三方编译器。假设对于同一个程序而言,3 个编译器生成的程序代码中指令总数相等,但是指令的组合情况各不相同。各类指令在 M1 和 M2 上运行时所需要的平均时钟周期数和在 3 类编译器生成的程序中所占的百分比(指令频度)如表 1.4 所示。

表 1.4 M1 和 M2 的 CPI 以及 3 个程序的指令频度

指令类	M1 的 CPI	M2 的 CPI	C1 的程序	C2 的程序	C3 的程序
A	2	1	30%	30%	50%
B	3	2	50%	20%	30%
C	4	1.5	20%	50%	20%

请回答下列问题:

(1) 如果 M1 和 M2 都使用 C1 编译器，则 M1 的生产厂商可以声称 M_1 的性能是 M2 的多少倍？

(2) 如果 M1 和 M2 都使用 C2 编译器，则 M2 的生产厂商可以声称 M_2 的性能是 M1 的多少倍？

(3) 如果你购买了 M1，那么你会选择哪种编译器？如果你购买了 M2，你又会选择哪种编译器？

(4) 如果其他所有指标(包括价格)都相同，你会买哪台机器？

【分析解答】

(1) 若 M1 和 M2 都用 C1 编译器，则 M1 的 CPI 为 2×30%+3×50%+4×20%=2.9，M2 的 CPI 为 1×30%+2×50%+1.5×20%=1.6；M1 的时钟频率为 800MHz，M2 的时钟频率为 400MHz，所以，M1 中一条指令的平均执行时间为 2.9×1/800MHz=3.625ns，M2 中一条指令的平均执行时间为 1.6×1/400MHz=4ns；M1 和 M2 的性能之比为 4ns/3.625ns=1.1，所以，M1 的生产商可以声称 M1 的性能是 M2 的 1.1 倍。

(2) 若 M1 和 M2 都用 C2 编译器，则 M1 的 CPI 为 2×30%+3×20%+4×50%=3.2，M2 的 CPI 为 1×30%+2×20%+1.5×50%=1.45；M1 的时钟频率为 800MHz，M2 的时钟频率为 400MHz，所以 M1 中一条指令的平均执行时间为 3.2×1/800MHz=4ns，M2 中一条指令的平均执行时间为 1.45×1/400MHz=3.625ns；M2 和 M1 的性能之比为 4ns/3.625ns=1.1，所以，M2 的生产商可以声称 M2 的性能是 M1 的 1.1 倍。

(3) 由上述分析可知：C1 编译器在 M1 上的性能比 M2 好，因此，如果购买了 M1，则编译器应选择 C1；同理，如果购买了 M2，则编译器应选择 C2。

(4) 从上述(1)和(2)分析结果来看，用生产商各自的编译器无法衡量 M1 和 M2 的好坏，借助第三方提供的编译器 C3 来考察更客观。若 M1 和 M2 都用 C3 编译器，则 M1 的 CPI 为 2×50%+3×30%+4×20%=2.7，M2 的 CPI 为 1×50%+2×30%+1.5×20%=1.4；M1 的时钟频率为 800MHz，M2 的时钟频率为 400MHz，所以 M1 中一条指令的平均执行时间为 2.7×1/800MHz=3.375ns，M2 中一条指令的平均执行时间为 1.4×1/400MHz=3.5ns；M1 和 M2 的性能之比为 3.5ns/3.375ns=1.04，用第三方编译器 C3 时 M1 的性能是 M2 的 1.04 倍，因此可选择购买 M1。

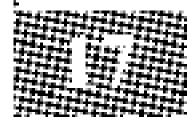
6. 若机器 M1 和 M2 具有相同的指令集，其时钟频率分别为 0.8GHz 和 1.6GHz。在指令集中有 5 种不同类型的指令 A～E。表 1.5 给出了在 M1 和 M2 中每类指令的平均时钟周期数 CPI。

表 1.5　在 M1 和 M2 中每类指令的平均时钟周期数 CPI

机器	A	B	C	D	E
M1	1	2	2	3	4
M2	2	2	4	5	6

请回答下列问题：

(1) M1 和 M2 的峰值 MIPS 各是多少？



(2) 假定程序P的指令序列中,5类指令具有完全相同的指令条数,则程序P在M1和M2中运行时,哪台机器更快?快多少?在M1和M2中执行程序P时的平均时钟周期数CPI各是多少?

【分析解答】

(1) 计算峰值MIPS时应该选择CPI最少的指令,故在M1中可以选择一段全部由A类指令组成的程序,其峰值MIPS为0.8GHz/1=800MIPS,在M2中可以选择一段全部由A类和B类指令组成的程序,其峰值MIPS为1.6GHz/2=800MIPS。

(2) 程序P中每类指令均占1/5,故M1的CPI为$CPI_1=(1+2+2+3+4)/5=2.4$,M2的CPI为$CPI_2=(2+2+4+5+6)/5=3.8$。当然,不能根据以上结果就说程序P在M1上运行更快,因为M1和M2的时钟频率不同。

假设程序P执行指令数为N,则P在M1上的执行时间为$2.4\times N\times 1/0.8=3.0N$ (ns);程序P在M2上的执行时间为$3.8\times N\times 1/1.6=2.375\ N$ (ns),所以,M2执行P的速度更快,每条指令平均快0.625ns。

从该题可以看出,虽然程序P在M1中每条指令执行所花的时钟周期数少,但是,因为M2的时钟频率更快,因而时钟周期更短,使得每条指令的平均执行时间更短。

7. 假设同一套指令集用不同的方法设计了两种机器M1和M2。机器M1的时钟周期为0.8ns,机器M2的时钟周期为1.2ns。某个程序P在机器M1上运行时的CPI为4,在M2上的CPI为2。对于程序P来说,哪台机器的执行速度更快?快多少?

【分析解答】

因为M1和M2实现的是同一套指令集,所以程序P在机器M1和M2上执行的指令条数相同,假定是N条,则P在M1上的执行时间为$4\times 0.8\text{ns}\times N=3.2N$ (ns);P在M2上的执行时间为$2\times 1.2\text{ns}\times N=2.4N$ (ns)。由此可知,对于程序P来说,M2的执行速度更快,平均每条指令快0.8ns。

8. 若机器M的时钟频率为2GHz,用户程序P在M上执行的指令条数为1×10^9,其CPI为1.5,则P在M上的执行时间是多少?若在机器M上从程序P开始启动到执行结束的延迟时间是2秒,则程序P的用户CPU时间占整个延迟时间的百分比是多少?

【分析解答】

程序P在机器M上执行所需时钟周期数为$1\times 10^9\times 1.5=1.5\times 10^9$,所需时间为$1.5\times 10^9/(2\times 10^9\text{Hz})=0.75\text{s}$,程序P的用户CPU时间在2秒钟的延时内占用的百分比为$0.75/2\times 100\%=37.5\%$。

9. 假定某编译器对某段高级语言程序编译生成两种不同的指令序列S1和S2,在时钟频率为1GHz的机器M上运行,目标指令序列中用到的指令类型有A、B、C和D共4类。4类指令在M上的CPI和执行两个指令序列所执行的各类指令条数如表1.6所示。

请问S1和S2各执行了多少条指令?CPI各为多少?所含的时钟周期数各为多少?执行时间各为多少?

表 1.6　4 类指令在 M 上的 CPI 和两个指令序列所执行的各类指令条数

	A	B	C	D
各指令的 CPI	1	2	3	4
S1 的指令条数	5	2	2	1
S2 的指令条数	1	1	1	5

【分析解答】

执行 S1 的指令条数为 5+2+2+1=10，CPI 为(5×1+2×2+2×3+1×4)/10=1.9，所含的时钟周期数为 1.9×10=19，故执行时间为 $19/(1\times10^9)$=19ns；

执行 S2 的指令条数为 1+1+1+5=8，CPI 为(1×1+1×2+1×3+5×4)/8=3.25，所含的时钟周期数为 3.25×8=26，故执行时间为 $26/(1\times10^9)$=26ns。

10. 假定机器 M 的时钟频率为 200MHz，程序 P 在机器 M 上的执行时间为 12 秒。对 P 优化时，将其所有的乘 4 指令都换成了一条左移两位的指令，得到优化后的程序 P′。若在 M 上乘法指令的 CPI 为 102，左移指令的 CPI 为 2，P 的执行时间是 P′执行时间的 1.2 倍，则 P 中有多少条乘法指令被替换成左移指令执行？

【分析解答】

P′的执行时间为 10 秒，P 比 P′多用了 2 秒钟，即多 200M×2=4×10^8 个时钟周期，每条乘法指令比左移指令多 100 个时钟周期，因为 $4\times10^8/100=4\times10^6$，所以有 400 万条乘法指令被替换为左移指令执行。

第 2 章 数据的机器级表示

2.1 教学目标和内容安排

主要教学目标：

使学生掌握计算机内部各种数据的机器级表示，并能将这些知识熟练运用到高级语言和机器级语言的编程和调试工作中。

基本学习要求：

(1) 了解真值和机器数的含义。

(2) 了解无符号整数的含义、用途和表示。

(3) 了解带符号整数的表示方法。

(4) 理解为什么现代计算机都用补码表示带符号整数。

(5) 掌握在真值和各种编码表示数之间进行转换的方法。

(6) 了解浮点数表示格式，及其与表示精度和表示范围之间的关系。

(7) 掌握规格化浮点数的概念和浮点数规格化方法。

(8) 掌握 IEEE 754 标准，并能在真值与单精度格式浮点数之间进行转换。

(9) 能运用数据表示知识解释和解决高级语言编程中数据表示和转换问题。

(10) 掌握常用的十进制数的二进制编码方法，如 8421 码。

(11) 了解逻辑数据、西文字符和汉字字符的常用表示方法，如 ASCII 码、GB 2312。

(12) 了解常用数据长度单位的含义，如 bit、Byte、KB、MB、GB、TB 等。

(13) 了解大端和小端排列方式，以及数据的对齐存储方式。

(14) 掌握奇偶校验和海明校验的基本原理。

(15) 掌握 CRC 码校验位的计算和检错方法。

本章内容相对比较容易，学生也比较熟悉，对于信息的二进制表示、进位计数制等简单内容，完全可以让学生自学。如果课时不充裕，对于十进制数的表示和汉字字符编码部分，也可以只简单介绍其概要内容，细节部分留给学生课后阅读。关于高级语言中的各种数据类型与机器级数据表示之间的关系，应该要求学生掌握，这对于提高学生程序设计和调试能力起到很大的作用。其实，这部分内容很简单，只要在教学过程中提醒学生关注并进行一些编程练习就能达到目的，而且程序设计课程中大多也会介绍这部分内容。

对于本章内容，教学过程中普遍存在的问题是，学生缺乏将机器级数据表示和程序设计

及程序调试工作相互关联的意识。许多学生也许对机器级数据表示的基本原理和概念很了解，但在程序设计和调试工作中，往往不会运用所学知识解决实际问题，不会把高级语言中的类型定义、数值范围、数据类型转换等问题和本课程所学的知识联系起来，因而，所学知识没有起到真正的作用。

为了增强学生对机器级数据表示的认识，可以让学生亲自编写相关的程序，通过程序的执行结果来理解本章所学的知识。与本章内容相关的编程练习可以有很多，以下给出一些示例。(1) 验证主教材[①]中表 2.2 给出的关系表达式的结果，并编程得出主教材第 2 章习题 8 表 2.14 中的结果；(2)确定 float 型变量和 double 型变量的精度；检查一些特殊表达式的运行结果，如一个非零整数除以 0、一个非零实数除以 0、0 除以 0、负数开平方等等；(3)检查机器是大端还是小端方式，数据是对齐存放还是不对齐存放。

2.2 主要内容提要

1. 数据的表示

主要有数值数据与非数值数据两类。

数值数据指在数轴上有对应的点，能比较大小的数，在计算机中有二进制数和十进制数两种表示形式。二进制表示有无符号整数、带符号整数和浮点数 3 类。无符号整数也称无符号数，用来表示地址等正整数；带符号整数一般用补码表示；浮点数用来表示实数，现代计算机中多采用 IEEE 754 标准。十进制表示的主要是整数，需要用二进制对其进行编码，因此也称为 BCD 码，最常用的 BCD 码是 8421 码。

非数值数据指在数轴上没有对应的点的数据。主要包括逻辑值、西文字符和汉字字符等。逻辑值只有两个状态取值，按位进行运算；西文字符多采用 7 位 ASCII 码表示；汉字字符有输入码、内码和字模码，汉字内码大多占 2 个字节。

2. 数据的宽度

通常以字节(Byte)为基本单位表示，数据长度单位(如 MB、GB、TB 等)在表示数据容量和带宽等不同对象时所代表的大小不同。

3. 数据的排列

有大端和小端两种排列方式。大端方式以 MSB 所在地址为数据的地址，即给定地址处存放的是数据最高有效字节；小端方式以 LSB 所在地址为数据的地址，即给定地址处存放的是数据最低有效字节。

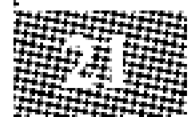
4. 数据校验方式

常用的数据校验方式有 3 种：奇偶校验、海明校验和循环冗余码校验。

奇偶校验方式根据数据的奇偶性变化来检错，只能检测奇数个错；海明校验采用分组奇偶校验，SEC 只能纠正一位错，SEC-DED 可纠正一位错并检测两位错；循环冗余码校验(CRC)通过某种数学运算在数据和校验位之间建立约定关系，它可以对较长数据块进行校验而不增加校验位开销，因此，主要用于对大批量数据的存储或传输校验。

---

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)



2.3 基本术语解释

机器数(Computer Word)

通常将数值数据在计算机内部编码表示的数称为机器数。机器数中只有 0 和 1 两种符号。

真值(Natural Number)

机器数真正的值(即原来带有正负号的数)称为机器数的真值。

数值数据(Numerical Data)

指有确定的值的数据,在数轴上能找到其对应的点,可以比较其大小。确定一个数值数据的值有 3 个要素:进位计数制、定/浮点表示和数的编码表示。也就是说,给定一个数字序列,如果不说明这个数字序列是几进制数,小数点的位置在哪里,采用什么编码方式,那么这个数字序列的值是无法确定的。

非数值数据(Non-numerical Data)

指在数轴上没有确定的值的数据,像逻辑数据、西文字符、汉字字符等都是非数值数据。

基数(Radix,Base)

进位计数制的"底数"或"基"。例如,二进制数的基数是"2",十进制数的基数为"10",十六进制的基数为"16"。

无符号整数(Unsigned Integer)

当一个编码的所有二进位都用来表示数值时,该编码表示的就是无符号整数,也称为无符号数,可以看成是正整数。常用于表示地址。

带符号整数(Signed Integer)

在计算机内部对正、负号进行了编码的整数。

定点数(Fixed-point Number)

定点数是计算机中小数点固定在最左边或最右边的数,有定点整数和定点小数两种。定点整数的小数点总是约定在数的最右边,主要用来表示现实世界中的整数和浮点数中的指数。定点小数的小数点总是约定在数的最左边,主要用来表示浮点数中的尾数。

定点数的编码方式有原码、反码、补码和移码;浮点数的尾数一般用原码小数来表示;浮点数的指数一般用移码来表示;而反码很少被使用,只用在某些特殊场合。

浮点数(Floating-point Number)

浮点数是可以指定小数点在不同位置的数。任意一个浮点数 F 可写成 $F=M\times 2^E$ 的形式。这样,一个浮点数就可以由两个定点数表示,M 称为浮点数的尾数(Mantissa,Significans),用一个定点小数来表示;E 称为浮点数的指数或阶码(Exponent),用一个定点整数来表示。

原码(Signed Magnitude)

由符号位直接跟数值位构成,也称"符号-数值"表示法。它的编码规则是:正号"+"用符号位"0"表示,负号"-" 用符号位"1"表示,数值部分不变。这种编码比较简单,但计算机处理不方便,20 世纪 50 年代以后,就不用它来表示整数了。现代计算机中,一般用它来表示浮点数的尾数,如 IEEE 754 标准就是这样。

反码(One's Complement)

一种对定点整数或定点小数进行二进制编码的编码方案。由于计算机处理反码没有补码方便,反码已很少被用了。

补码(Two's Complement)

补码编码规则是:正号"+"用符号位"0"表示,负号"-"用符号位"1"表示,正数的数值部分不变,负数的数值部分是"各位取反,末位加1"。这种编码较原码复杂,但由于它是一种模运算系统,计算机处理很方便。常用补码表示带符号整数。

变形补码(Four's Complement)

变型补码是一种双符号位补码,又称为"模4-补码"。双符号位可以用来检测定点整数是否发生溢出,左符号位为真正的符号位,右符号位用来判别是否溢出。采用"变形补码"进行溢出检测时的判断规则为:"当结果的两个符号位不同时,发生溢出"。双符号位通常用于保存运算过程中进到高位的数值部分。

移码(Excess Notation,Biased Notation)

移码编码规则是:将真值加上一个偏置常数(Bias)。因为在浮点数的加减运算中,要进行对阶操作,需要比较两个阶的大小。用移码表示阶码后,使得所有数的阶码都相当于一个正整数,比较大小时,就只要按高位到低位顺序比较就行了,因而,移码主要用来表示浮点数的阶码,可以简化阶码的比较过程。

单精度浮点数(Single Precision Floating Point)

指IEEE 754标准规定的32位浮点数格式表示的浮点数。阶码用8位移码表示,偏置常数为127,尾数用原码表示,规格化浮点数的最高位"1"隐含不表示,显式表示的尾数有23位,所以一共有24位尾数。

双精度浮点数(Double Precision Floating Point)

指IEEE 754标准规定的64位浮点数格式表示的浮点数。阶码用11位移码表示,偏置常数为1023,尾数用原码表示,规格化浮点数的最高位"1"隐含不表示,显式表示的尾数有52位,所以一共有53位尾数。

机器零(Machine "0")

用一种专门的位序列表示"机器0"。例如,IEEE 754单精度浮点数中,用"0000 0000H"表示"+0",用"8000 0000H"表示"-0"。当运算结果出现阶码过小时,计算机将该数近似表示为"机器0"。

BCD码(Binary Coded Decimal,BCD)

十进制数用二进制编码的形式表示称为BCD码。

逻辑数据(Logic Data)

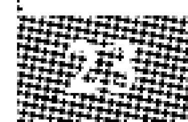
逻辑数据用来表示命题的"真"和"假",分别用"1"和"0"来表示。进行逻辑运算时,按位进行。

ASCII码(American Standard Code for Information Interchange)

目前计算机中使用最广泛的西文字符集及其编码,即美国标准信息交换码(American Standard Code for Information Interchange),简称ASCII码。

汉字输入码(Chinese Character Input Code)

对每个汉字用一个标准键盘上按键的组合来表示的编码方式。一般分为数字编码(如



区位码)、字音编码(如微软拼音、全拼)、字形编码(如五笔字型)和形音编码。

汉字内码(Chinese Character Code)

是指在计算机内部进行汉字存储、查找、传输和处理而采用的编码方式,通常用 2 个字节表示一个汉字内码。

机器字长(Machine Word Length)

一个二进制位(bit,比特)是计算机内部信息表示的最小单位。而机器字长指的是特定计算机中 CPU 用于整数运算的数据通路的宽度,通常也就是 CPU 内定点运算器和通用寄存器的二进制位数。

编址单位(Addressing Unit)

指对主存单元编号时具有相同编号的二进制位数。主存单元的编号称为地址。通常的编址单位为 8,即字节,称为按字节编址。按字编址时,编址单位为字。

字地址(Word Address)

按字节编址时,一个字可能占用几个内存单元,字地址就是这几个连续内存单元地址中的最小值。

最高有效位(Most Significant Bit,MSB)

一个二进制数中的最高位。例如二进制数 1000 中的"1"。

最低有效位(Least Significant Bit,LSB)

一个二进制数中的最低位。例如二进制数 1110 中的"0"。

最高有效字节(Most Significant Byte,MSB)

一个二进制数中的最高字节。例如二进制数 1111 1111 0000 0000 1111 0000 中的 1111 1111。

最低有效字节(Least Significant Byte,LSB)

一个二进制数中的最低字节。例如二进制数 1111 1111 0000 0000 1111 0000 中的 1111 0000。

大端方式(Big Endian)

采用字节编址方式时,一个多字节数据(如 int、float 等类型数据)将占用多个主存单元。大端方式下,将数据字的最低有效字节 LSB 存放在大地址单元中,即字地址是 MSB 所在单元的地址。

小端方式(Little Endian)

采用字节编址方式时,一个多字节数据(如 int、float 等类型数据)将占用多个主存单元。小端方式下,将数据字的最低有效字节 LSB 存放在小地址单元中,即字地址是 LSB 所在单元的地址。

边界对齐(Boundary Alignment)

要求数据的地址是相应的边界地址。例如,按字节编址时,4 字节长数据的地址应该是 4 的倍数,即最末两位总是 00,2 字节长数据的地址总是 2 的倍数。

检错和纠错(Error Detect and Correct)

数据在计算机内部被处理的过程中会因为硬件故障而产生差错。因此需要采用数据校验方法来发现是否有错,即检错。有些数据校验方法可以准确发现错误的位置从而可以将二进制值取反,即纠错。

码距(Code Distance)

两个码字中具有不同代码的位数叫码字间的“距离”。在一种编码方式下所有可能出现的码字中,任意两两进行比较得到一组距离,其中的最小值称为这种编码方式的“码距”。例如,8421 码的码距为 1。

奇偶校验(Parity Check)

在原二进制数据中增加一个奇校验位(偶校验位),使“1”的个数为奇数(偶数),最后在该数据传输的结果中检查“1”的个数是否保持为奇数(偶数),如是,则认为正确或有偶数个位置出错,否则,认为有奇数个位置出错。

海明码(Hamming Codes)

主要用于存储器中的数据校验,将数据按某种规律分成若干组,对每组进行相应的奇偶检测。最终比较时,按位进行异或操作,根据异或操作结果,确定在哪一位发生了差错。

循环冗余校验(Cyclic Redundancy Check,CRC)

常用于外存储器或数据通信中的数据校验,检错纠错能力强。编码时通过某种数学运算根据原数据生成校验位,在检查时对含校验位的数据进行相应的数学运算,根据运算结果即可完成检错和纠错。

2.4 常见问题解答

1. 真值和机器数的关系是什么?

答: 在计算机内部用二进制编码表示的数称为机器数,而机器数真正的值(即原来带有正负号的数)称为机器数的真值,所以,它们之间的关系就是同一个数据的两种不同表示形式。

2. 什么是编码?

答: 编码是指用少量简单的基本符号,对大量复杂多样的信息进行一定规律的组合。基本符号的种类和组合规则是信息编码的两大要素。例如,用 10 个阿拉伯数字表示数值,电报码中用 4 位十进制数字表示汉字,等等,都是编码的典型例子。计算机内部处理的所有信息都是用 0 和 1 编码了的信息。

3. 什么是数字化编码?

答: “数字化编码”就是对感觉媒体信息(如数值、文字、图像、声音、视频等信息)进行定时采样,将现实世界中的连续信息转换为计算机中离散的“样本”信息,然后对这些离散的“样本”信息用 0 和 1 进行二进制编码。

4. 计算机内部为什么用二进制来编码所有信息?

答: 主要有 3 个方面的原因:

(1) 二进制系统只有两个基本符号:“0”和“1”。所以,它的基本符号少,易于用稳态电路实现。

(2) 二进制的编码/计数/运算等规则简单。

(3) 二进制中的“0”和“1”与逻辑命题的“真”和“假”的对应关系简单,为计算机中实现逻辑运算和程序中的逻辑判断提供了便利的条件,特别是能通过逻辑门电路方便地实现算术运算。

5. 计算机内都用二进制表示信息，为什么还要引入八进制和十六进制？

答：计算机内部在进行信息的存储、传送和运算时，都是以二进制形式来表示信息的。但在屏幕上或书本上书写信息时，由于二进制信息位数多，阅读、记忆不方便，而十六进制、八进制和二进制的对应关系简单，又便于阅读、记忆和书写，所以引入十六进制或八进制，使得人们在开发、调试程序和阅读机器内部代码时，能方便地用八进制或十六进制来等价表示二进制信息。

6. 如何表示一个数值数据？计算机中的数值数据都是二进制数吗？

答：在计算机内部，数值数据的表示方法有两大类：

(1) 直接用二进制数表示。分为无符号数和有符号数，有符号数又分为定点数表示和浮点数表示。无符号数用来表示无符号整数(如地址等信息)；定点数用来表示整数；浮点数用来表示实数。

(2) 采用二进制编码的十进制数(Binary Coded Decimal Number，BCD码)来表示整数，BCD码的编码方案有很多，但一般都采用8421码(也称为NBCD码)来表示。

因此，计算机中的数值数据虽然都用二进制来编码表示，但不全是二进制数，也有用十进制数表示的。因而有些处理器的指令类型中，就有对应的二进制加法指令和十进制加法指令。

7. 为什么要引入无符号数表示？

答：因为有些情况下只要对正整数进行运算，且结果不出现负值，此时，可以用无符号数表示变量。例如，在进行地址或指针运算时可用无符号数。

8. 在高级语言程序中定义的unsigned型数据是怎么表示的？

答：unsigned型数据就是无符号数，直接用二进制对数值进行编码得到的就是无符号数。

9. 为什么无符号数运算时结果可能会发生"溢出"？什么叫无符号数的"溢出"？

答：计算机的机器字长总是有限的，因而机器数的位数有限，使得可表示的数的个数有限。对于n位二进制数，只能表示2^n个不同的数，当运算结果超过n位数时就可能发生溢出。

对于无符号数来说，计算机运算过程中一般保留低n位，舍弃高位。这样，会产生两种结果：

(1) 剩下的低n位数不能正确表示运算结果。这种情况下，意味着运算的结果超出了计算机能表达的范围，有效数值进到了第$n+1$位，称此时发生了"溢出"现象。例如，对于4位无符号数相加运算，当计算$14+3$时就会发生溢出，即$1110+0011=10001$，结果中第一位1是数值部分，这个1丢弃后结果就不对了。

(2) 剩下的低n位数能正确表达计算结果，也即高位的舍去并不影响其运算结果。例如，对于4位无符号数相减运算，当计算$14-3$时，用14加-3的补码来实现，即$1110+1101=11011$，结果中第一位1丢弃后，结果是正确的。

"对一个多于n位的数丢弃高位而保留低n位数"这样一种处理，实际上等价于"将这个多于n位的数去除以2^n，然后丢去商保留其余数"的操作。这种操作运算就是"模运算"。在一个模运算系统中，运算的结果最终都是丢弃高位而保留低位。所以，只要不是"溢出"(即只要真正的值不会进到第$n+1$位)，结果就是正确的。这是模运算系统的特点。

10. 为什么现代计算机都用补码来表示整数?

答：和原码、反码相比，用补码表示定点整数时，有以下4个好处：(1)符号位可以和数值位一起参加运算；(2)补码可以实现模运算，即可用加法方便地实现减法运算；(3)零的表示唯一；(4)可以多表示一个最小负数。所以，现代计算机中都采用补码来表示定点整数。

11. n 位二进制补码整数的模是多少? 数的表示范围是什么?

答：n 位二进制补码整数的模是 2^n，表示其运算结果只保留低 n 位，多于 n 位的高位部分取模后要被丢弃，其数值范围为 $-2^{(n-1)} \sim +2^{(n-1)}-1$。

12. 在高级语言编程时定义的short/int/long型数据是怎么表示的?

答：int型数据就是定点整数，现代计算机一般用补码来表示。int型数据的位数与运行平台和编译器有关，一般是32位或16位。long型数据和short型数据也都是定点整数，用补码表示，只是位数不同，分别是长整型和短整型数。

13. 在C语言程序中，关系表达式"－2147483648＝＝2147483648U"的结果为什么为"真"?

答：关系表达式"－2147483648＝＝2147483648U"的左边是负数，右边是正数，因此，左右两数看似不等，结果似乎应该为"假"。但是，C语言中，如果执行一个运算时同时有无符号整数(如unsigned int)和带符号整数(如int)参加，那么，C编译器会隐含地将带符号整数强制类型转换为无符号整数，因此，在上面的关系表达式运算中，左边的带符号整数"－2147483648"对应的机器数"1000 0000 0000 0000 0000 0000 0000 0000"被解释成无符号数，其值为 2^{31}，和右边的无符号数"2147483648U"的值完全相同，因而结果为"真"。

14. 定点整数在数轴上分布的点之间都是等距的吗?

答：是的。定点整数在数轴上的点总是在整数值上，即[…，－5，－4，－3，－2，－1，0，1，2，3，4，5，…]，相邻数据间隔总是1。

15. 为什么要引入浮点数表示?

答：因为定点数不能表示实数，而且表数范围小，所以，要引入浮点数表示。

16. 为什么浮点数的阶(指数)要用移码表示?

答：因为在浮点数的加减运算中，要进行对阶操作，需要比较两个阶的大小。移码表示的实质就是把阶加上一个偏置常数，使得所有数的阶码都转换为一个正整数，比较大小时，就只要按高位到低位顺序比较，因而，引入移码可以简化阶的比较过程。

17. 浮点数如何表示0?

答：用一种专门的位序列表示0，例如，IEEE 754单精度浮点数中，用"0000 0000H"表示"＋0"，用"8000 0000H"表示"－0"。当运算结果出现阶码过小时，计算机将该数近似表示为0，称为"机器0"。

18. 现代计算机中采用什么标准来表示浮点数?

答：早期的计算机各自采用不同的浮点数表示格式，因而，在不同计算机之间进行数据交换时，就会发生数据格式不统一的问题。所以，专门制定了IEEE 754标准用来规定计算机中的浮点数表示格式。现代计算机中都采用IEEE 754标准来表示浮点数。

19. 为什么浮点数要采用规格化形式表示?

答：为了使浮点数中能尽量多地表示有效位数，提高浮点数运算的精度。

20. 如何判断一个浮点数是否是规格化数？

答：只要看转换为真值后，其尾数的第一位是否一定是非零数。因此，对于原码编码的尾数来说，只要看尾数的数值部分第一位是否为1；对于补码表示的尾数，只要看符号位和尾数最高位是否相反。

21. 浮点数表示的精度和数值范围取决于什么？

答：浮点数的精度取决于尾数的位数，而数值范围取决于阶码的位数。在浮点数总位数不变的情况下，阶码位数越多，则尾数位数越少。即表数范围越大，则精度越差（数变稀疏）。

22. 基数的大小对表数范围和精度有什么影响？

答：基数越大，则范围越大，但精度变低（数变稀疏）。

23. 在高级语言编程中，float 和 double 型数据是怎么表示的？

答：现代计算机用 IEEE 754 标准表示浮点数，其中 32 位单精度浮点数就是 float 型数据，64 位双精度浮点数就是 double 型数据。

24. 在高级语言编程中，long double 型数据是怎么表示的？

答：long double 型数据的长度和格式随编译器和处理器类型的不同而有所不同。例如，Microsoft Visual C++ 6.0 版本以下的编译器都不支持该类型，因此，用其编译出来的目标代码中 long double 和 double 一样，都是 64 位双精度；在 IA-32 上使用 gcc 编译器时，long double 类型数据采用 Intel x86 FPU 的 80 位双精度扩展格式（1 位符号位 s、15 位阶码 e、1 位显式首位有效位(Explicit Leading Significand Bit) j 和 63 位尾数 f）表示；在 SPARC 和 PowerPC 处理器上使用 gcc 编译器时，long double 类型数据采用相应的 128 位双精度扩展格式（1 位符号位 s、15 位阶码 e 和 112 位尾数 f，采用隐藏位，故有效位数为 113 位）表示。

25. C 语言程序中，为什么关系表达式"123456789＝＝(int)(float)123456789"的结果为"假"，而关系表达式"123456＝＝(int)(float)123456"和"123456789＝＝(int)(double)123456789"的结果都为"真"？

答：首先应该明白，在 C 语言中，float 类型对应 IEEE 754 单精度浮点数格式，也即 float 型数据的有效位数只有 24 位（相当于有 7 位十进制有效位数）；double 类型对应 IEEE 754 双精度浮点数格式，有效位数有 53 位（相当于有 17 位十进制有效位数）；int 类型为 32 位整数，有 31 位有效位数（最大数为 2147483647）。

整数 123456789 的有效位数为 9 位，转换为 float 型数据后肯定发生了有效位数丢失，再转换为 int 型数据时，已经不是 123456789 了，所以，等式"123456789＝(int)(float)123456789"两边的值不相等。

数据改为 123456 后，有效位数只有 6 位，转换为 float 型数据后有效位数没有丢失，因而数值没变，再转换为 int 型数据时，还是 123456，所以，等式"123456＝＝(int)(float)123456"两边的值相等。

整数 123456789 的有效位数为 9 位，转换为 double 型数据后，不会发生有效位数丢失，再转换为 int 型数据时，还是 123456789，所以，等式"123456789＝＝(int)(double)123456789"两边的值相等。

26. 位数相同的定点数和浮点数中，可表示的浮点数个数比定点数个数多吗？

答：不是的。可表示的数据个数取决于编码所采用的位数。编码位数一定，则编码出

的数据个数就是一定的。n 位编码最多只能表示 2^n 个数，所以，对于相同位数的定点数和浮点数来说，可表示的数据个数应该一样多。但是，有时由于一个值可能有两个或多个编码对应，编码个数会有少量差异。

27. 如何进行 BCD 码的编码？

答：每位十进制数的取值可以是 0/1/2/…/9 这十个数之一，因此，每一个十进制数位必须至少有 4 位二进制位来表示。而 4 位二进制位可以组合成 16 种状态，去掉 10 种状态后还有 6 种冗余状态，所以从 16 种状态中选取 10 种状态表示十进制数位 0～9 的方法有很多，可以产生多种 BCD 码方案。大的方面可分为有权码和无权码两种。

有权码指表示每个十进制数位的 4 个二进制数位(称为基 2 码)都有一个确定的权。8421 码是最常用的十进制有权码；无权码指表示每个十进制数位的 4 个基 2 码没有确定的权。

28. 逻辑数据在计算机中如何表示？如何运算？

答：逻辑数据用来表示命题的“真”和“假”，分别用“1”和“0”来表示。进行逻辑运算时，按位进行。

29. 汉字的区位码、国标码和机内码有什么区别？

答：GB 2312 字符集由 94 行、94 列组成，行号称为区号，列号称为位号，各占 7 位，共 14 位，区号在左、位号在右，称为汉字的区位码，它指出了该汉字在码表中的位置。

汉字的国标码是将区号、位号各加上 32(即十六进制的 20H)后，再在前后 7 位前各加 0。

汉字的内码需 2 个字节才能表示，可以在国标码的基础上产生汉字机内码，一般是将国标码的两个字节的第一位设置成“1”。

例如，已知一个汉字的国标码为 343AH，前后两个字节各减 32(20H)得到区位码为 343AH－2020H＝141AH，所以区号为 20(14H)，位号为 26(1AH)；机内码是将国标码的两个字节的最前面一位变为“1”，因此，机内码为 B4BAH。

30. MSB(LSB)表示最高(低)有效字节还是最高(低)有效位？

答：MSB 的含义可能是最高有效字节(Most Significant Byte)，也可能是最高有效位(Most Significant Bit)，具体表示哪一个含义要看上下文。同样，LSB 的含义可能是最低有效字节(Least Significant Byte)，也可能是最低有效位(Least Significant Bit)。

31. 有时用“字”表示数据的宽度，一个“字”到底有多少位？

答：除了用“比特(Bit)”和“字节(Byte)”来表示一个数据的宽度外，有时也用“字(Word)”来表示数据宽度的单位。不同的计算机，其“字”的长度和组成不完全相同，有的由 2 个字节组成，有的由 4 个、8 个甚至 16 个字节组成。

32. 一个“字”的宽度就是一个“机器字长”吗？

答：不是。“机器字长”是计算机的一个非常重要的指标。通常称 32 位机器或 64 位机器，就是指机器的字长是 32 位或 64 位。一般情况下，机器字长定义为 CPU 中一次能够处理的二进制整数的位数，实际上就是 CPU 中整数运算数据通路的位数。

“字”作为信息宽度的计量单位，对于某个系列机来说，其字宽总是固定的。例如，在 80x86 系列中，一个字的宽度为 16 位，因此，32 位是双字，64 位是四字。在 IBM 303X 系列中，一个字的宽度为 32 位，所以 16 位为半字，32 位为单字，64 位为双字。

一个“字”的宽度可以不等于机器字长。例如，在 Intel 微处理器中，从 80386 开始就都

是32位或以上的机器了，即机器字长至少为32位，但其字的宽度都定义为16位。

33. 在表示存储容量和带宽时经常用到KB、MB、GB、TB等表示数据量的单位，为什么有时1MB等于10^6字节，有时又等于2^{20}字节呢？

答：当表示二进制存储容量时，度量单位用2的幂次，例如，若主存容量为1GB，则表示主存有2^{30}字节。当描述距离、频率等数值时，通常用10的幂次表示，因而在由时钟频率计算得到的总线带宽或外设数据传输率中，度量单位表示的也是10的幂次。例如，若总线带宽为1GB/s，表示总线每秒传输10^9字节。为区分这种差别，通常用K表示1024，用k表示1000，而其他前缀字母均为大写，表示的大小由其上下文决定。

2.5 单项选择题

1. 计算机中的所有信息都以二进制表示的原因是（　　）。
   A. 信息处理方便　　B. 运算速度快
   C. 节约元器件　　D. 物理器件特性所致
2. 引入八进制和十六进制的目的是（　　）。
   A. 节约元件
   B. 实现方便
   C. 可以表示更大范围的数
   D. 用于等价地表示二进制，便于阅读和书写
3. 108对应的十六进制形式是（　　）。
   A. 6CH　　B. B4H　　C. 5CH　　D. 63H
4. 下列数中最小的数为（　　）。
   A. $(1001\ 0110)_2$　　B. $(63)_8$
   C. $(1001\ 0110)_{BCD}$　　D. $(2F)_{16}$
5. 下列数中最小的数为（　　）。
   A. $(1110\ 0101)_2$　　B. $(93)_{10}$
   C. $(1001\ 0010)_{BCD}$　　D. $(5A)_{16}$
6. 负零的补码表示为（　　）。
   A. 1 00…00　　B. 0 00…00　　C. 0 11…11　　D. 1 11…11
7. $[x]_补=x_0.x_1x_2\cdots x_n$（$n$为整数），它的模是（　　）。
   A. 2^{n-1}　　B. 2^n　　C. 1　　D. 2
8. $[x]_补=x_0x_1x_2\cdots x_n$（$n$为整数），它的模是（　　）。
   A. 2^{n+1}　　B. 2^n　　C. 2^n+1　　D. 2^n-1
9. 下列编码中，零的表示形式是唯一的编码是（　　）。
   A. 反码　　B. 原码　　C. 补码　　D. 原码和补码
10. 在下列有关补码和移码关系的叙述中，错误的是（　　）。
    A. 相同位数的补码和移码表示具有相同的表数范围
    B. 零的补码和移码表示相同
    C. 同一个数的补码和移码表示，其数值部分相同，而符号位相反

D. 一般用移码表示浮点数的阶，而补码表示定点整数

11. 计算机内部的带符号整数大多用补码表示，以下是一些关于补码特点的叙述：

① 零的表示是唯一的

② 符号位可以和数值部分一起参加运算

③ 和其真值的对应关系简单、直观

④ 减法可用加法来实现

以上叙述中，哪些选项是补码表示的特点？（　　）

A. ①、②　　B. ①、③　　C. ①、②、③　　D. ①、②、④

12. 假定某数 $x=-0100\ 1010B$，在计算机内部的表示为 1011 0110B，则该数所用的编码方法是（　　）。

A. 原码　　B. 反码　　C. 补码　　D. 移码

13. 设寄存器位数为 8 位，机器数采用补码形式（含一位符号位），则十进制数 -26 存放在寄存器中的内容为（　　）。

A. 26H　　B. 9BH　　C. E6H　　D. 5AH

14. -1029 的 16 位补码用十六进制表示为（　　）。

A. 0405H　　B. 7BFBH　　C. 8405H　　D. FBFBH

15. 考虑以下 C 语言代码：

```
short si=-8196;
unsigned short usi=si;
```

执行上述程序段后，usi 的值是（　　）。

A. 8196　　B. 34572　　C. 57339　　D. 57340

16. 设$[x]_{原}=1.x_1x_2x_3x_4$，当满足（　　）时，$x>-1/2$ 成立。

A. x_1 必须为 1，$x_2x_3x_4$ 至少有一个为 1

B. x_1 必须为 1，$x_2x_3x_4$ 任意

C. x_1 必须为 0，$x_2x_3x_4$ 至少有一个为 1

D. x_1 必须为 0，$x_2x_3x_4$ 任意

17. 设 $x=-0.1011B$，则 8 位补码$[x]_{补}$为（　　）。

A. 1.101 1000B　　B. 1.000 1011B

C. 1.010 1000B　　D. 1.000 0101B

18. 16 位无符号数所能表示的数值范围是（　　）。

A. $0\sim(2^{16}-1)$　　B. $0\sim(2^{15}-1)$

C. $0\sim2^{16}$　　D. $0\sim2^{15}$

19. 16 位补码整数所能表示的范围是（　　）。

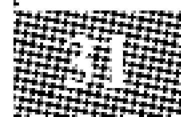
A. $-2^{15}\sim+(2^{15}-1)$　　B. $-(2^{15}-1)\sim+(2^{15}-1)$

C. $-2^{16}\sim+(2^{16}-1)$　　D. $-(2^{16}-1)\sim+(2^{16}-1)$

20. 若浮点数尾数用补码表示，则下列数中为规格化尾数形式的是（　　）。

A. 1.110 0000B　　B. 0.011 1000B

C. 0.010 1000B　　D. 1.000 1000B



21. 若浮点数尾数用原码表示，则下列数中为规格化尾数形式的是(　　)。

A. 1.110 0000B　　B. 0.011 1000B

C. 0.01 01000B　　D. 1.000 1000B

22. 用于表示浮点数阶码的编码通常是(　　)。

A. 原码　　B. 补码　　C. 反码　　D. 移码

23. 若某数采用 IEEE 754 单精度浮点数格式表示为 4510 0000H，则其值是(　　)。

A. $(+1.125)_{10}\times 2^{10}$　　B. $(+1.125)_{10}\times 2^{11}$

C. $(+0.125)_{10}\times 2^{11}$　　D. $(+0.125)_{10}\times 2^{10}$

24. 若某数采用 IEEE 754 单精度浮点数格式表示为 C820 0000H，则其值是(　　)。

A. $(-1.01)_{10}\times 2^{17}$　　B. $(-1.01)_{10}\times 2^{144}$

C. $(-1.25)_{10}\times 2^{17}$　　D. $(-1.25)_{10}\times 2^{144}$

25. 假定变量 i、f 的数据类型分别是 int、float。已知 $i=12345$，$f=1.2345e3$，则在一个 32 位机器中执行下列表达式时，结果为“假”的是(　　)。

A. $i==(int)(float)i$　　B. $i==(int)(double)i$

C. $f==(float)(int)f$　　D. $f==(float)(double)f$

26. IBM 370 的短浮点数格式中，总位数为 32 位，左边第一位(b_0)为数符，随后 7 位($b_1\sim b_7$)为阶码，用移码表示，偏置常数为 64，右边 24 位($b_8\sim b_{31}$)为 6 位十六进制原码小数表示的尾数，采用规格化形式。若将十进制数－265.625 用该浮点数格式表示，则应表示为(　　)。(用十六进制形式表示)

A. C310 9A00H　　B. 4310 9A00H

C. 8310 9A00H　　D. 0310 9A00H

27. 假定两种浮点数表示格式的位数都是 32 位，但格式 1 的阶码长，尾数短，而格式 2 的阶码短，尾数长，其他所有规定都相同，则它们可表示的数的精度和范围为(　　)。

A. 两者可表示的数的范围和精度均相同

B. 格式 1 可表示的数的范围更小，但精度更高

C. 格式 2 可表示的数的范围更小，但精度更高

D. 格式 1 可表示的数的范围更大，且精度更高

28. 在一般的计算机系统中，西文字符编码普遍采用(　　)。

A. BCD 码　　B. ASCII 码

C. 格雷码　　D. CRC 码

29. 假定某计算机按字节编址，采用小端方式，有一个 float 型变量 x 的地址为 FFFF C000H，x=1234 5678H，则在内存单元 FFFF C001H 中存放的内容是(　　)。

A. 1234H　　B. 34H

C. 56H　　D. 5678H

30. 下面有关机器字长的叙述中，错误的是(　　)。

A. 机器字长是指 CPU 中定点运算数据通路的宽度

B. 机器字长一般与 CPU 中寄存器的位数有关

C. 机器字长决定了数的表示范围和表示精度

D. 机器字长对计算机硬件的造价没有影响

31. 下面是关于计算机中存储器容量单位的叙述，其中错误的是(　　)。
A. 最小的计量单位为位(bit)，表示一位“0”或“1”
B. 最基本的计量单位是字节(Byte)，一个字节等于 8b
C. 一台计算机的编址单位、指令字长和数据字长都一样，且是字节的整数倍
D. 主存容量为 1KB，其含义是主存中能存放 1024 个字节的二进制信息

32. 假定下列字符编码中含有奇偶检验位，但没有发生数据错误，那么采用奇校验的字符编码是(　　)。
A. 0101 0011　　B. 0110 0110
C. 1011 0000　　D. 0011 0101

33. 假设需要传送的一个数据块的长度为 10 位，对每个数据块采用 CRC 校验，若约定的生成多项式为 x^3+1，则对应的 CRC 码的位数是(　　)。
A. 3　　B. 4　　C. 13　　D. 14

34. 假设某个需要校验的数据为 0110 0101B，采用 CRC 校验，生成多项式为 x^4+x+1，则对应的校验码是(　　)。
A. 0010　　B. 0110　　C. 1110　　D. 1111

【参考答案】

1. D	2. D	3. A	4. D	5. D	6. B	7. D
8. A	9. C	10. B	11. D	12. C	13. C	14. D
15. D	16. D	17. C	18. A	19. A	20. D	21. A
22. D	23. B	24. C	25. C	26. A	27. C	28. B
29. C	30. D	31. C	32. C	33. C	34. A	

2.6　分析应用题

1. 实现下列各数的转换。
(1) $(25.8125)_{10}=(?)_2=(?)_8=(?)_{16}$
(2) $(101101.011)_2=(?)_{10}=(?)_8=(?)_{16}=(?)_{8421}$
(3) $(0101\ 1001\ 0110.0011)_{8421}=(?)_{10}=(?)_2=(?)_{16}$
(4) $(4E.C)_{16}=(?)_{10}=(?)_2$

【分析解答】
(1) $(25.8125)_{10}=(11001.1101)_2=(31.64)_8=(19.D)_{16}$
(2) $(101101.011)_2=(45.375)_{10}=(55.3)_8=(2D.6)_{16}=(0100\ 0101.0011\ 0111\ 0101)_{8421}$
(3) $(0101\ 1001\ 0110.0011)_{8421}=(596.3)_{10}=(1001010100.0100)_2=(254.4)_{16}$
(4) $(4E.C)_{16}=(78.75)_{10}=(1001110.11)_2$

2. 假定机器数为 8 位(1 位符号，7 位数值)，写出下列各二进制数的原码和补码表示。
+0.1001，−0.1001，+1.0，−1.0，+0.010100，−0.010100，+0，−0

【分析解答】
上述各二进制数的原码和补码表示在表 2.1 中给出。

表 2.1 小数的原码和补码表示

数 值	原 码	补 码
+0.1001	0.1001000	0.1001000
−0.1001	1.1001000	1.0111000
+1.0	溢出	溢出
−1.0	溢出	1.0000000
+0.010100	0.0101000	0.0101000
−0.010100	1.0101000	1.1011000
+0	0.0000000	0.0000000
−0	1.0000000	0.0000000

3. 假定机器数为8位(1位符号,7位数值),写出下列各二进制数的补码和移码表示。
+1001,−1001,+1,−1,+10100,−10100,+0,−0

【分析解答】

上述各二进制数的补码和移码表示在表2.2中给出。

表 2.2 整数的补码和移码表示

数 值	补 码	移码(偏置常数=1 0000000)
+1001	0 0001001	1 0001001
−1001	1 1110111	0 1110111
+1	0 0000001	1 0000001
−1	1 1111111	0 1111111
+10100	0 0010100	1 0010100
−10100	1 1101100	0 1101100
+0	0 0000000	1 0000000
−0	0 0000000	1 0000000

4. 设$[x]_{补}=1.x_1x_2x_3x_4$,当$x_1x_2x_3x_4$满足什么条件时,$x<-1/2$成立?

【分析解答】

补码的编码规则是:“正数的补码,其符号位为0,数值位不变;负数的补码,其符号位为1,数值位各位取反,末尾加1”。从形式上来看,$[x]_{补}$的符号位为1,故x一定是负数。因此,绝对值越大,数值越小,因而要满足$x<-1/2$,则x的绝对值必须大于1/2。因此,x_1必须为0,$x_2x_3x_4$至少有一个为1,这样,各位取反末尾加1后,x_1一定为1,$x_2x_3x_4$中至少有一个为1,使得x的绝对值保证大于1/2。因此,x_1必须为0,$x_2x_3x_4$至少有一个为1。

5. 已知$[x]_{补}$,求x。

(1) $[x]_{补}=1.1100001B$　　(2) $[x]_{补}=1\ 0000000B$

(3) $[x]_{补}=0.1111111B$　　　　(4) $[x]_{补}=1\ 1111111B$

【分析解答】

(1) $x=-0.0011111B$　　　　(2) $x=-10000000B=-128$

(3) $x=0.1111111B$　　　　(4) $x=-00000001B=-1$

6. 将以下十进制数表示成无符号整数时至少需要几个二进位?

156,820,1200,4503

【分析解答】

$2^7-1<156<2^8-1$,故至少需要 8 位。

$2^9-1<820<2^{10}-1$,故至少需要 10 位。

$2^{10}-1<1200<2^{11}-1$,故至少需要 11 位。

$2^{12}-1<4503<2^{13}-1$,故至少需要 13 位。

7. 假定某程序中定义了 3 个变量 x、y 和 z,其中 x 和 z 为 int 型,y 为 short 型。当 $x=-258$,$y=-20$ 时,执行赋值语句 $z=x-y$ 后,存放 z 的寄存器中的内容是多少?

【分析解答】

现代计算机中带符号整数都是用补码表示的,因此,本题可以直接计算 z 的值,然后将 z 的补码形式求出来,也可以先将 x 和 y 的补码求出,再通过补码加法求出 z 的补码表示。显然,前一种思路效率较高。对于前一种思路,执行赋值语句后,$z=-238$,因此,问题就变成了求-238 的补码表示,其结果为$[-000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1110\ 1110B]_{补}=1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0001\ 0010B=FFFF\ FF12H$。

8. 假定 sizeof (int)=4,表 2.3 中第一列给出了 C 语言程序中的关系表达式,请参照已有表栏内容完成表中后 3 栏内容的填写,并对关系表达式"2147483647 < (int) 2147483648U"的结果进行说明。

表 2.3　关系表达式的运算结果

关系表达式	运算类型	结果	说　明
0==0U			
−1<0			
−1<0U	无符号整数	0	11…1B($2^{32}-1$)>00…0B(0)
2147483647>−2147483647−1	有符号整数	1	011…1B($2^{31}-1$)>100…0B (-2^{31})
2147483647U>−2147483647−1			
2147483647<(int)2147483648U			
−1>−2			
(unsigned)−1>−2			

【分析解答】

按照题目要求填表 2.4:

表 2.4　与表 2.3 对应的关系表达式的运算结果

关系表达式	运算类型	结果	说　　明
0==0U	无符号整数	1	00…0B=00…0B
−1<0	有符号整数	1	11…1B(−1)<00…0B(0)
−1<0U	无符号整数	0	11…1B ($2^{32}-1$)>00…0B(0)
2147483647>−2147483647−1	有符号整数	1	011…1B($2^{31}-1$)>100…0B(-2^{31})
2147483647U>−2147483647−1	无符号整数	0	011…1B($2^{31}-1$)<100…0B(2^{31})
2147483647<(int)2147483648U	有符号整数	0	011…1B($2^{31}-1$)>100…0B(-2^{31})
−1>−2	有符号整数	1	11…1B(−1)>11…10B(−2)
(unsigned)−1>−2	无符号整数	1	11…1B ($2^{32}-1$)>11…10B($2^{32}-2$)

8 个关系表达式运算结果分别是 1、1、0、1、0、0、1、1，其中 1 表示“真”，0 表示“假”。关系表达式“2147483647 < (int) 2147483648U”的结果为“假”。因为小于号右边的“2147483648U”是一个带后缀 U 的整数，因而是无符号整数，其机器数为“100…0”(1 后面跟 31 个 0)，其值为 2^{31}。强制类型转换为 int 类型后，其真值为 -2^{31}，即“−2147483648”，显然“2147483647 <−2147483648”是不成立的，也即结果为“假”。

9. 以下是一个 C 语言程序，用来计算一个数组 *a* 中每个元素的和。当参数 *len* 为 0 时，返回值应该是 0，但在执行时，却发生了存储器访问异常。请问这是什么原因造成的，并说明程序应该如何修改。

```
1    float sum_elements (float a[], unsigned len)
2    {
3        int    i;
4        float  result=0;
5
6        for (i=0; i<=len-1; i++)
7            result+=a[i];
8        return result;
9    }
```

【分析解答】

存储器访问异常是由于对数组 *a* 访问时产生了越界错误而造成的。循环变量 *i* 是 int 型，而 *len* 是 unsigned 型，当 *len* 为 0 时，执行 *len*−1 的结果为 32 个 1，是最大可表示的 32 位无符号数，任何无符号数都比它小，使得循环体被不断执行，导致数组访问越界，因而发生存储器访问异常。应当将参数 *len* 声明为 int 型。

10. 下列几种情况所能表示的数的范围是什么？

(1) 16 位无符号整数

(2) 16 位原码定点小数

(3) 16 位补码定点小数

(4) 16 位补码定点整数

(5) 下述格式的浮点数(基数为 2,移码的偏置常数为 128,规格化尾数,不考虑隐藏位)

数符	阶码	尾数
1位	8位移码	7位原码

【分析解答】

(1) 16 位无符号整数范围为 $0 \sim 2^{16}-1$,即 0～65535。

(2) 16 位原码定点小数表示的范围为 $-(1-2^{-15}) \sim +(1-2^{-15})$。

(3) 16 位补码定点小数表示的范围为 $-1 \sim +(1-2^{-15})$。

(4) 16 位补码定点整数表示的范围为 $-2^{15} \sim +(2^{15}-1)$,即 $-32768 \sim +32767$。

(5) 规格化浮点数的表示范围如下。

最大正数:$+0.1111111B \times 2^{1111\ 1111B} = +(1-2^{-7}) \times 2^{127}$

最小正数:$+0.1000000B \times 2^{0000\ 0000B} = +2^{-1} \times 2^{-128} = +2^{-129}$

最大负数:$-0.1000000B \times 2^{0000\ 0000B} = -2^{-1} \times 2^{-128} = -2^{-129}$

最小负数:$-0.1111111B \times 2^{1111\ 1111B} = -(1-2^{-7}) \times 2^{127}$

由于原码是关于原点对称的,所以,浮点数的表示范围是关于原点对称的。

对于非规格化浮点数,最小正数和最大负数的尾数形式为±0.000 0001B,因而值为 $\pm 2^{-7} \times 2^{-128} = \pm 2^{-135}$。

11. 设某浮点数格式为:

数符	阶码	尾数
1位	5位移码	6位补码

其中,移码的偏置常数为 16,补码采用一位符号位,基数为 4,规格化尾数,不考虑隐藏位。

(1) 用这种格式表示下列十进制数:+1.625,−0.125,+20,−9/16。

(2) 写出该格式浮点数的表示范围。

【分析解答】

(1) $+1.625 = +1.1010B = (+0.122)_4 \times 4^1$,故阶码为 1+16=17=10001B,尾数为四进制数+0.122 的补码,即 0.011010B,因此,+1.625 表示为 0 10001 011010B。

$-0.125 = -0.0010B = (-0.200)_4 \times 4^{-1}$,故阶码为 −1+16=15=01111B,尾数为四进制数−0.200 的补码,即 1.100000B,因此,−0.125 表示为 1 01111 100000B。

$+20 = +10100B = (+0.110)_4 \times 4^3$,故阶码为 3+16=19=10011B,尾数为四进制数+0.110 的补码,即 0.010100B,因此,+20 表示为 0 10011 010100B。

$-9/16 = -0.1001B = (-0.210)_4 \times 4^0$,故阶码为 0+16=16=10000B,尾数为四进制数−0.210 的补码,即 1.011100B,因此,−9/16 表示为 1 10000 011100B。

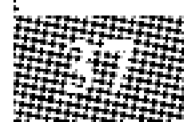
(2) 规格化浮点数的表示范围如下。

最大正数:$+0.111111B \times 4^{11111B} = (+0.333)_4 \times 4^{15}$

最小正数:$+0.010000B \times 4^{00000B} = (+0.100)_4 \times 4^{-16} = +4^{-17}$

最大负数:$-0.010000B \times 4^{00000B} = (-0.100)_4 \times 4^{-16} = -4^{-17}$

最小负数:$-1.000000B \times 4^{11111B} = (-1.000)_4 \times 4^{15} = -4^{15}$



由于补码表示的尾数不是关于原点对称的，所以，浮点数的表示范围不是相对于原点对称的。

12. 以 IEEE 754 单精度浮点数格式表示下列十进制数，要求结果写成十六进制形式。

+1.625，-0.125，+20，-9/16

【分析解答】

$+1.625=+1.101B\times 2^0$，所以，符号 $s=0$，阶码 $e=0+127=0111\ 1111B$，尾数的小数部分 $f=0.101B$，因此，+1.625 用 IEEE 754 单精度浮点数格式表示为 0011 1111 1101 0000 0000 0000 0000 0000B，用十六进制形式表示为 3FD0 0000H。

$-0.125=-0.001B=-1.0B\times 2^{-3}$，所以，符号 $s=1$，阶码 $e=-3+127=0111\ 1100B$，尾数的小数部分 $f=0.0B$，因此，-0.125 用 IEEE 754 单精度浮点数格式表示为 1011 1110 0000 0000 0000 0000 0000 0000B，用十六进制形式表示为 BE00 0000H。

$+20=+10100B=+1.01B\times 2^4$，所以，符号 $s=0$，阶码 $e=4+127=1000\ 0011B$，尾数的小数部分 $f=0.01B$，因此，+20 用 IEEE 754 单精度浮点数格式表示为 0100 0001 1010 0000 0000 0000 0000 0000B，用十六进制形式表示为 41A0 0000H。

$-9/16=-0.1001B=-1.001B\times 2^{-1}$，所以，符号 $s=1$，阶码 $e=-1+127=0111\ 1110B$，尾数的小数部分 $f=0.001B$，因此，-9/16 用 IEEE 754 单精度浮点数格式表示为 1011 1111 0001 0000 0000 0000 0000 0000B，用十六进制形式表示为 BF10 0000H。

13. 假定一个 float 型变量 x 的机器数为 4510 0000H，则变量 x 的值是多少？

【分析解答】

float 型变量的机器数对应 IEEE 754 单精度浮点数格式，因此，将 4510 0000H 展开为 32 位机器数：0100 0101 0001 0000 0000 0000 0000 0000 后，得到其符号为 0；阶码为 1000 1010B-127=11；尾数的值为 1.001B=1.125。因而 x 的数值为 $+1.125\times 2^{11}=2304$。

14. 设一个变量的值为 2049，要求分别用 32 位补码整数和 IEEE 754 单精度浮点格式表示该变量（结果用十六进制表示），并说明哪段二进制序列在两种表示中完全相同，为什么会相同？

【分析解答】

$2049=1000\ 0000\ 0001B=+1.000\ 0000\ 0001B\times 2^{11}$，用 32 位补码整数表示为 0000 0000 0000 0000 0000 **1000 0000 0001**，用十六进制形式表示为 0000 0801H；用 IEEE 754 单精度浮点数格式表示时，符号 $s=0$，阶码 $e=11+127=10001010B$，尾数的小数部分 $f=0.000\ 0000\ 0001B$，因此，2049 用 IEEE 754 单精度浮点数格式表示为 0 100 0101 0**000 0000 0001** 0000 0000 0000B，用十六进制形式表示为 4500 1000H。

在上述两种表示中，存在相同的序列 000 0000 0001。因为 2049 被转换为规格化浮点数后，有效数值部分中最前面的 1 被隐藏，剩余部分为 000 0000 0001，而 2049 的 32 位补码整数表示中保留了完整的有效数值部分，即最前面的 1 没有被隐藏，所以除了这个 1 之外后面的二进制序列 000 0000 0001 是相同的。

15. 设一个变量的值为 -2147483646，要求分别用 32 位补码整数和 IEEE 754 单精度浮点格式表示该变量（结果用十六进制表示），并说明哪种表示其值完全精确，哪种表示的是近似值。

【分析解答】

$-2147483646=-111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110B=-1.11\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110B\times 2^{30}$，32位补码形式为1000 0000 0000 0000 0000 0000 0000 0010B（8000 0002H），IEEE 754单精度格式为1100 1110 1111 1111 1111 1111 1111 1111B（CEFF FFFFH），因为−2147483646在$-2^{31}\sim 2^{31}-1$范围内，所以可用32位补码精确表示；对于IEEE 754单精度浮点数格式，最多只可表示24位有效二进位数字，而−2147483646的有效二进位有30位，后面的有效二进位必须截断，因而是近似表示。

16. 假定变量 i 和 f 的数据类型分别为 int 和 float，sizeof（int）＝4，已知 $i=1234567890$，$f=1.23456789e10$，要求给出以下各关系表达式的结果，并说明原因。

(1) $i==$(int)(float)i　　　　(2) $i==$(int)(double)i

(3) $f==$(float)(int)f　　　　(4) $f==$(float)(double)f

【分析解答】

(1) 结果为“假”。因为 float 类型采用 IEEE 754 单精度浮点数格式，尾数的小数部分只有23个二进位和一位隐藏位，共有24位有效位数，相应地，十进制有效位数为7位，而 i 中有9位有效位数，因而将 i 转换为 float 类型时会发生有效数字的丢失，再转换为 int 类型时，其值已经被改变了。

(2) 结果为“真”。因为 double 类型采用 IEEE 754 双精度浮点数格式，其有效位数为 $52+1=53$ 个二进位，而 int 类型的有效位数有31个二进位，因而，对于任何一个 int 类型的变量，转换为 double 类型数据后，精度不会有任何损失，再转换回 int 类型时，其值不变。

(3) 结果为“假”。因为变量 f 的值超过了 int 类型可表示的最大值，因而将 f 转换为 int 类型后再转换回 float 类型时，其值已经改变。

(4) 结果是“真”。因为 double 类型的精度比 float 类型高，任何 float 类型变量的值转换为 double 后再转换回 float 类型时，其值不变。

17. 假定某32位字长的机器中带符号整数用补码表示，浮点数用 IEEE 754 标准表示，寄存器 R1 和 R2 的内容分别为 8020 0000H 和 0080 0000H。若执行下列运算指令时，操作数为寄存器 R1 和 R2 的内容，则 R1 和 R2 中操作数的真值分别为多少？

(1) 无符号数加法指令

(2) 带符号整数乘法指令

(3) 单精度浮点数减法指令

【分析解答】

不同指令对寄存器内容进行不同的操作，因而，不同指令执行时寄存器内容对应的真值不同。

(1) 对于无符号数加法指令，R1 和 R2 的内容均被解释成无符号整数，即 R1 的真值为 8020 0000H，R2 的真值为 80 0000H，也即 R1 的真值为 $2^{31}+2^{21}$，R2 的真值为 2^{23}。

(2) 对于带符号整数乘法指令，R1 和 R2 的内容均被解释为补码整数，由最高位可知，R1 为负数，R2 为正数。R1 的真值为 −0111 1111 1110 0000 0000 0000 0000 0000B，即 −7FE0 0000H；R2 的真值为 +80 0000H，也即 R1 的真值为 $-(2^{31}-2^{21})$，R2 的真值为 2^{23}。

(3) 对于单精度浮点数减法指令，R1 和 R2 的内容均为 IEEE 754 单精度浮点数。由 R1 的内容可知，其符号位为1，表示负数，阶码为 0000 0000B，尾数部分为 010 0000 0000

0000 0000 0000B，因为阶码为全 0，尾数为非 0 数，故 R1 是非规格化浮点数，其指数为 -126，尾数为 0.01B，故 R1 表示的真值为 $-0.01B\times2^{-126}=-2^{-128}$。由 R2 的内容可知，其符号位为 0，表示正数，阶码为 0000 0001B，尾数部分为 000 0000 0000 0000 0000 0000B，R1 为规格化浮点数，其指数为 $1-127=-126$，尾数为 1.0B，故 R2 表示的真值为 $+1.0B\times2^{-126}=2^{-126}$。

18. IBM 370 的短浮点数格式中，总位数为 32 位，左边第一位（b_0）为数符，随后 7 位（$b_1\sim b_7$）为阶码，用移码表示，偏置常数为 64，右边 24 位（$b_8\sim b_{31}$）为 6 位十六进制原码小数表示的尾数，采用规格化形式，基为 16。若将十进制数 -260.125 用该浮点数格式表示，则对应的机器数是什么？（要求用十六进制形式表示）

【分析解答】

IBM 370 的短浮点数格式的尾数采用十六进制原码表示，基数是 16。因此，在进行数据转换时，要先转化成十六进制形式。即 $-260.125=-0001\ 0000\ 0100.0010B=(-104.2)_{16}=(-0.1042)_{16}\times16^3$。由此可知，浮点数符号位应为 1，指数为 3，用 7 位移码表示为 $64+3=$ 100 0011B，故前 8 位表示为 1 100 0011B，对应的十六进制为 C3H，尾数部分的 6 位十六进制数为 10 4200H，因此，对应的机器数为 C310 4200H。

19. 1991 年 2 月 25 日，海湾战争中，美国在沙特阿拉伯的达摩地区设置的爱国者导弹拦截伊拉克的飞毛腿导弹失败，致使飞毛腿导弹击中了美国的一个兵营，造成了 28 名士兵死亡。拦截失败的原因是由于一个浮点数的精度问题造成的。爱国者导弹系统中有一个内置时钟，用计数器实现，每隔 0.1 秒计数一次。程序用 0.1 乘以计数器的值得到以秒为单位的时间。0.1 的二进制表示是一个无限循环序列：0.00011[0011]B（方括号中的序列是重复的）。请问：

（1）假定用一个类型为 float 的变量 x 来表示 0.1，则变量 x 在机器中的机器数是什么（要求写成十六进制形式）？绝对值 $|x-0.1|$ 的值是什么（要求用十进制表示）？

（2）爱国者系统启动时计数器的初始值为 0，并开始持续计数。假定当时系统运行了 200 个小时，则程序计算的时间和实际时间的偏差为多少？如果爱国者根据飞毛腿的速度乘以它被侦测到的时间来预测位置，若飞毛腿的速度为 2000 米/秒，则预测偏差的距离为多少？

【分析解答】

（1）0.1=0.0 0011[0011]B=+1.1 0011 0011 0011 0011 0011 00B$\times2^{-4}$，float 类型采用 IEEE 754 单精度浮点数格式。符号位 s 为 0，阶码 $e=127-4=123=$0111 1011B，尾数的小数部分为 0.1001 1001 1001 1001 1001 100B，因此，在机器中 float 型变量 x 表示为 0 011 1101 1 100 1100 1100 1100 1100 1100B，用十六进制形式表示为 3DCC CCCCH。由于 float 类型的精度有限，只有 24 位有效位数，尾数从最前面的 1 开始一共只能表示 24 位，后面的有效数字全部被截断，故 x 与 0.1 之间的误差为 $|x-0.1|$=0.000 0000 0000 0000 0000 0000 0000 1100 [1100]B。这个值约等于 $0.11B\times2^{-27}$，大约为 5.59×10^{-9}。

（2）爱国者系统运行 200 个小时后，共计数 $200\times60\times60\times10=72\times10^5$ 次。因此，程序计算的时间和实际时间的偏差约为 $5.59\times10^{-9}\times72\times10^5=0.0402$s。预测偏差距离约为 2000m/s$\times$0.0402s=80.4m。

20. 假定浮点数的阶码用 m 位移码表示，偏置常数为 $2^{m-1}-1$，规格化尾数的整数部分为 1，是隐藏位，小数部分有 n 位，用原码表示，基为 2，请回答下列问题。

(1) 能用这种浮点数格式精确表示的最小正整数是多少？

(2) 不能用这种浮点数格式精确表示的最小正整数是多少？

【分析解答】

(1) 能用这种浮点数格式表示的最小正整数为1。

(2) 这种浮点数格式的有效位数为 $n+1$ 位，因此，当某个正整数的有效位数大于 $n+1$ 位时，则 $n+1$ 位后的有效数字被截断，也即不能用这种浮点数格式精确表示。因此，不能用这种浮点数格式表示的最小正整数为+10…01B(中间有 n 位0)，其值为 $2^{n+1}+1$。

21. 图2.1(a)是某个C++程序，图2.1(b)是该程序的若干组执行结果。请根据IEEE 754标准的舍入规定对执行结果进行解释说明，并通过分析得出float变量的有效位数。

```
#include <iostream>
using namespace std;
int  main()
{
   float    heads;
   cout.setf(ios::fixed,ios::floatfield);
   while(1)
   {
   cout<<"please  enter  a  number:";
   cin>>heads;
   cout<<heads<<endl;
   }
   return 0;
}
```

(a) C++源程序

```
Please enter a nmber: 61.419997
61.419998
Please enter a number: 61.419998
61.419998
Please enter a number: 61.419999
61.419998
Please enter a number: 61.42
61.419998
Please enter a number: 61.420001
61.420002
Please enter a number:
```

(b) 程序运行结果

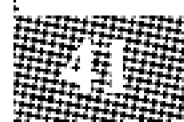
图2.1 源程序及其运行结果示例

【分析解答】

该程序的功能非常简单，就是从键盘输入一个实数，赋给一个float型变量后再从屏幕上输出。从运行结果来看，61.419998和61.420002是两个可表示数，两者之间相差0.000004。当输入数据是一个不可表示数时，机器将其转换为最邻近的可表示数。

目前几乎所有机器中float型变量都是采用IEEE 754单精度浮点数格式表示，其二进制有效位数为24位，因此相应的十进制有效位数为7位。因为61=111101B=1.11101B×2^5，如果将float型数据的规格化正数的表示范围以 2^i($-126\leqslant i\leqslant 127$)为分割点划分成若干区间，61.419998应该位于区间[2^5,2^6]，该区间相邻可表示数之间的间隔为 $2^{-23}\times 2^5=2^{-18}=0.0000038\cdots\approx 0.000004$，从上述分析结果来看，该区间相邻两个可表示数之间的间隔就是0.000004。因此，在61.419998前面的可表示数为61.419994，后面的可表示数为61.420002。

显然，当输入61.419997和61.419999时，其最靠近的可表示数为61.419998；而61.420001的最邻近可表示数为61.420002；当输入为61.42时，从十进制形式61.42000来看，它位于可表示数61.419998和61.420002的中点，但是，实际上机器内部是按照二进制



表示来判断的，从二进制表示形式来看，61.42 应该更靠近 61.419998，因此，61.42 对应输出的可表示数为 61.419998。

22. 假定在一个程序中定义了变量 x、y 和 i，其中，x 和 y 是 float 型变量(用 IEEE 754 单精度浮点数表示)，i 是 16 位 short 型变量(用补码表示)。程序执行到某一时刻，$x=-130$、$y=7.25$、$i=130$，它们都被写到了主存(按字节编址)中，其地址分别是 $\&x$，$\&y$ 和 $\&i$。请分别给出在大端机器和小端机器上变量 x、y 和 i 在内存的存放位置。

【分析解答】

$x=-130=-10000010B=-1.000001B\times2^7$，阶码 $e=127+7=128+6=1000\ 0110B$，所以，用 IEEE 754 单精度浮点数表示为 1100 0011 0000 0010 0000 0000 0000 0000B=C302 0000H。

$y=7.25=111.01B=+1.1101B\times2^2$，阶码 $e=127+2=128+1=1000\ 0001B$，所以，用 IEEE 754 单精度浮点数表示为 0100 0000 1110 1000 0000 0000 0000 0000B=40E8 0000H。

$i=130=1000\ 0010B$，用 16 位补码表示为 0082H。

上述 3 个数据在大端机器和小端机器上的存放位置如表 2.5 所示。

表 2.5 数据在大端机器和小端机器中的存放位置

地址	大端机器	小端机器
$\&x$	C3H	00H
$\&x+1$	02H	00H
$\&x+2$	00H	02H
$\&x+3$	00H	C3H
$\&y$	40H	00H
$\&y+1$	E8H	00H
$\&y+2$	00H	E8H
$\&y+3$	00H	40H
$\&i$	00H	82H
$\&i+1$	82H	00H

23. 假定某计算机存储器按字节编址，CPU 从存储器中读出一个 4 字节信息 D=3234 3538H，该信息的内存地址为 0000 F00CH，按小端方式存放，请回答下列问题。

(1) 该信息 D 占用了几个内存单元？这几个内存单元的地址及其内容各是什么？

(2) 若 D 是一个 32 位无符号数，则其值是多少？

(3) 若 D 是一个 32 位补码表示的带符号整数，则其值是多少？

(4) 若 D 是一个 IEEE 754 单精度浮点数，则其值是多少？

(5) 若 D 是一个用 8421 码表示的无符号整数，则其值是多少？

(6) 若 D 是一个字符串，每个字节的低 7 位表示对应字符的 ASCII 码，则对应字符串是什么？

(7) 若D是两个汉字的国标码,则这两个汉字在GB 2312字符集码表中分别位于哪一行和哪一列?

(8) 若D中前3个字节分别是一个像素的R、G、B分量的颜色值,则其值各是多少?

【分析解答】

将3234 3538H展开为二进制表示为0011 0010 0011 0100 0011 0101 0011 1000B。

(1) 因为存储器按字节编址,所以4个字节占用4个内存单元,其地址分别是0000 F00CH、0000 F00DH、0000 F00EH、0000 F00FH。由于采用小端方式存放,所以,最低有效字节38H存放在0000 F00CH中,35H存放在0000 F00DH中,34H存放在0000 F00EH中,32H存放在0000 F00FH中。

(2) 无符号数:其值为$2^{29}+2^{28}+2^{25}+2^{21}+2^{20}+2^{18}+2^{13}+2^{12}+2^{10}+2^{8}+2^{5}+2^{4}+2^{3}$。

(3) 补码整数:符号为0,表示其为正数,其值为与无符号数的值一样。

(4) IEEE 754单精度浮点数:根据IEEE 754单精度浮点数格式可知,符号位$s=0$,为正数;阶码$e=0110\ 0100B=100$,值为$100-127=-27$;尾数小数部分$f=0.\ 011\ 0100\ 0011\ 0101\ 0011\ 1000B$,所以,其值为$1.011\ 0100\ 0011\ 0101\ 0011\ 1B\times 2^{-27}$。

(5) 8421码整数:表示对应十进制数32343538。

(6) ASCII码字符串:各字节的低7位分别为011 0010B、011 0100B、011 0101B、011 1000B,所以,对应的字符串为"2458"。

(7) 汉字:对国标码每个字节各自减20H,得到两个汉字的区位码分别为1214H和1518H,即,第一个汉字在GB 2312字符集码表中位于第18(12H)行、第20(14H)列,第二个汉字位于第21(15H)行、第24(18H)列。

(8) 颜色值:R、G、B分量的颜色值分别为0011 0010B=50,0011 0100B=52,0011 0101B=53。

24. 已知下列字符编码:A=100 0001B,a=110 0001B,0=011 0000B,求D、d、6的7位ACSII码和第一位前加入奇校验位后的8位编码。

【分析解答】

D的ASCII码为100 0001B+011B=100 0100B,前面加奇校验位后的编码是1 100 0100B。

d的ASCII码为110 0001B+011B=110 0100B,前面加奇校验位后的编码是0 110 0100B。

6的ASCII码为011 0000B+110B=011 0110B,前面加奇校验位后的编码是1 011 0110B。

25. 某数据为0110 0101B,采用奇校验和CRC校验(生成多项式为x^4+x+1)对应的校验码各是什么?

【分析解答】

采用奇校验时,因为数据中有偶数个1,故校验位为1;采用CRC校验时,生成多项式为x^4+x+1,故有4个校验位。首先在数据后面添4个0,得到0110 0101 0000B,再将它和10011进行模2除法,得到余数为0010B,所以校验码为0010B。

26. 假定某计算机的总线采用偶校验，每8位数据有一位校验位，若在32位数据线上传输的信息为1234 5678H，则对应的4个校验位应为什么？若接收方收到的数据信息和校验位分别为1235 5678H和0100B，则说明发生了什么情况，给出验证过程。

【分析解答】

总线上传输的4个字节分别为0001 0010B，0011 0100B，0101 0110B，0111 1000B，因此对应的4个偶校验位P_0～P_3分别为0，1，0，0。当接收方收到数据信息1235 5678H和校验位0100B后，进行如下验证。

第1字节：数据为12H＝0001 0010B，故校验位$P_0'=0,0\oplus0=0$，说明传输正确。

第2字节：数据为35H＝0011 0101B，故校验位$P_1'=0,0\oplus1=1$，说明传输错误。

第3字节：数据为56H＝0101 0110B，故校验位$P_2'=0,0\oplus0=0$，说明传输正确。

第4字节：数据为78H＝0111 1000B，故校验位$P_3'=0,0\oplus0=0$，说明传输正确。

27. 假定一个16位数据$M_{16}M_{15}M_{14}M_{13}M_{12}M_{11}M_{10}M_9M_8M_7M_6M_5M_4M_3M_2M_1$为0111 1000 1101 0010B，要求写出16位数据的SEC码，并说明SEC码如何正确检测数据位M_5的错误。

【分析解答】

对于16位数据的单检错码，其校验位P和故障字S都是5位。假定故障字$S=S_5S_4S_3S_2S_1$的值反映对应数据位或校验位出错，则得到码字排列和故障字的取值如下。

	M_{16}	M_{15}	M_{14}	M_{13}	M_{12}	P_5	M_{11}	M_{10}	M_9	M_8	M_7	M_6	M_5	P_4	M_4	M_3	M_2	P_3	M_1	P_2	P_1
S_5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_4	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
S_3	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
S_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
S_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

根据上表，得到各个校验位的计算公式如下：

$$P_1=M_1\oplus M_2\oplus M_3\oplus M_4\oplus M_5\oplus M_7\oplus M_9\oplus M_{11}\oplus M_{12}\oplus M_{14}\oplus M_{16}$$

$$P_2=M_1\oplus M_3\oplus M_4\oplus M_6\oplus M_7\oplus M_{10}\oplus M_{11}\oplus M_{13}\oplus M_{14}$$

$$P_3=M_2\oplus M_3\oplus M_4\oplus M_8\oplus M_9\oplus M_{10}\oplus M_{11}\oplus M_{15}\oplus M_{16}$$

$$P_4=M_5\oplus M_6\oplus M_7\oplus M_8\oplus M_9\oplus M_{10}\oplus M_{11}$$

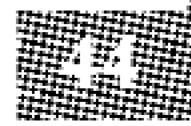
$$P_5=M_{12}\oplus M_{13}\oplus M_{14}\oplus M_{15}\oplus M_{16}$$

当数据字为0111 1000 1101 0010B时，代入上述公式，得到校验位$P=P_5\ P_4\ P_3\ P_2\ P_1=$ 00011B。第5位数据M_5出错，数据字变为0111 1000 1100 0010B，代入上述公式得到$P'=P_5'P_4'P_3'P_2'P_1'=01010B$，因而，故障字$S=S_5S_4S_3S_2S_1=00011B\oplus01010B=01001B$，说明码字中第9位($M_5$)出错。

28. 假设要传送的数据信息为111 0001B，若约定的生成多项式为$G(x)=x^3+1$，则校验码为多少？假定在接收端接收到的数据信息为111 0011B，说明如何正确检测其错误，写出检测过程。

【分析解答】

原数据信息为111 0001B，对应的报文多项式为$M(x)=x^6+x^5+x^4+1$，因为生成多项



式的位数为 4 位，所以校验码的位数为 3 位。在原数据信息后面添加 3 个 0，得到 $M'(x)=x^3M(x)=x^9+x^8+x^7+x^3$，用 $M'(x)$去模 2 除 $G(x)$，得到余数 110B，因此，在发送端要传送的 CRC 码为 111 0001 110B。假定 CRC 码中的校验位在传送过程中没有出错，在接收端还是 110B，在接收端收到的数据位为 111 0011B，则在接收端对数据校验时，将接收端的 CRC 码 111 0011 110B 用生成多项式 $G(x)$进行模 2 除，得到的余数为 010B。余数不为 0 表明传输时发生了错误。

第 3 章 运算方法和运算部件

3.1 教学目标和内容安排

主要教学目标：

使学生掌握核心运算部件 ALU 以及计算机内部各种基本运算算法和运算部件的相关知识，能够运用所学知识分析和解释高级语言和机器级语言程序设计中遇到的各种问题和相应的执行结果。

基本学习要求：

(1) 了解高级程序设计语言和低级程序设计语言中涉及的各种运算。

(2) 掌握定点数的逻辑移位、算术移位和扩展操作方法。

(3) 了解原码加减运算的基本原理。

(4) 掌握补码加减运算方法，并能设计补码加减运算器。

(5) 了解定点数乘法和除法运算的基本思想。

(6) 了解专用的阵列乘法器和阵列除法器的基本思想。

(7) 理解为何在运算中会发生溢出，并掌握各类定点数运算的溢出判断方法。

(8) 掌握浮点数加减运算的过程和方法。

(9) 理解 IEEE 754 标准对附加位的添加以及舍入模式等方面的规定。

(10) 了解浮点数乘法和除法运算的基本思想。

(11) 掌握算术逻辑单元 ALU 的功能和结构。

(12) 了解串行加法器和快速并行加法器的基本工作原理。

(13) 了解定点运算器(即定点运算数据通路)的基本结构。

(14) 了解浮点数加减运算器的基本结构。

本章涉及各种类型数据的各种运算算法和运算电路，因此内容多而烦琐。特别是原码加减运算和各种类型的乘除运算，它们的基本原理都很简单，但实现起来非常复杂。在课堂教学中，这些内容往往占用很多课时，而且烦琐的步骤和一些算法规定也经常使学生感觉枯燥无味。这些内容在本课程内容框架体系中，不属于主干内容，对这些内容掌握得好坏与深浅程度基本不会影响学生对其他知识的学习。因此，对于原码加减运算，只要根据现实世界中十进制加减运算规则去理解，把原理讲清楚就行了，没有必要让学生死记硬背运算规则；对于乘法运算，只要将原码一位乘法和补码一位乘法(布斯乘法)的基本原理、两位一乘算法

的基本思想，以及阵列乘法器的基本思想讲清楚就行了。对于除法运算，只要将原码除法运算的基本原理，补码除法运算的特点，以及阵列除法器的基本思想讲清楚就行了。对于乘除运算的学习，其主要目的不是学会怎样模拟计算机进行乘除运算，而是能够认识到乘除运算的算法复杂性和相对较大的时间开销，并且认识到不同实现方法所用时间开销是不同的，这种不同主要是由于运算部件控制方式的不同而造成的。

课时有限的情况下，有关原码加减运算、两位乘法运算、补码除法运算、阵列乘法器、阵列除法器、浮点数乘除运算、定点运算部件、浮点运算部件等加"*"的部分，都可以只用一两句话概括讲一下基本思想或提出问题以引起学生进一步思考并留作课后阅读，无须占用大量的课堂时间来介绍细节内容。

为了增强学生对计算机内部各类运算算法的认识，可以让学生亲自编写相关的程序，通过程序的执行结果来理解本章所学的知识。与本章内容相关的编程练习示例如下。(1)给定一组无符号数和有符号整数变量的值，对其进行移位操作后，查看其结果，并进行分析解释；(2)对于某个负数，如－4098，将该数赋值给一个 short 类型变量，然后将其转换为 unsigned short、int、unsigned int、float 等类型，查看其结果，并进行分析解释；(3)对于某个大的正整数，如 2147483647($2^{31}-1$)，将该数赋值给某个 int 类型变量，再将其转换为 short、unsigned short、unsigned int、float 等类型，查看其结果，并进行分析解释；(4)对于某个十进制有效位数多于 8 的实数，如 123456.789e5，将其定义为 float 型变量，然后转换为 double 型变量；再反过来将 double 型转换为 float 型，查看其结果，并进行分析解释；(5)对于各类运算，给出一些特殊的例子，以进行"溢出"、"大数吃小数"等方面的验证，并分析结果以给出合理的解释。例如，对于无符号数（如 unsigned int），计算"1＋4294967295"、"1－4294967295"的值。对于带符号整数（如 int），计算"2147483647＋1"、"－2147483648－1"的值。对于浮点数，分别计算"(1.0＋123456.789e30)＋(－123456.789e30)"和"1.0＋(123456.789e30＋(－123456.789e30))"的值，并查看两个结果是否一致，等等。

3.2　主要内容提要

1. 加法器

根据进位方式的不同，有 3 种基本加法器：行波进位加法器、进位选择加法器和先行（超前）进位加法器。行波进位加法器将多个一位全加器串行连接，各进位串行传递，速度慢；进位选择加法器通过选择两个分别带进位 0 和 1 的高位部分加法器的输出来实现高、低两部分的并行执行，使运算时间减半；先行（超前）进位加法器通过"进位生成"和"进位传递"函数来使各进位独立、并行产生，速度快。可用单级、两级或更多级先行进位方式连接。采用先行进位方式能加快加法器速度，目前多用这种方式。

2. 算术逻辑单元(ALU)

在先行进位加法器的基础上增加其他逻辑可构成 ALU，以实现基本的加/减算术运算和基本逻辑运算。ALU 的输入有：两个操作数、一位低位来的进位、一组操作控制信号，ALU 的输出有：一个结果、一位向高位的进位，以及零标志(ZF)和溢出标志(OF)等。

3. 定点运算及定点运算器

定点运算由专门的定点运算器实现，其核心部件是带先行进位加法器的 ALU，在控制

逻辑的控制下，可进行各种逻辑运算和算术运算。除基本的与或非等逻辑运算外，主要的运算包括以下几种。

(1) 移位运算：包括逻辑移位、算术移位和循环移位。逻辑移位对无符号数进行，移位时，在空出的位补0，左移时可根据移出位是否为1来判断溢出；算术移位对带符号整数进行，移位前后符号位保持不变，否则溢出；循环移位时不需要考虑溢出。左移一位，数值扩大一倍，相当于乘2操作；右移一位，数值缩小一半，相当于除2操作。

(2) 扩展运算：包括零扩展和符号扩展。零扩展对无符号数进行，高位补0；符号扩展对带符号整数进行，因为用补码表示，所以在高位直接补符号。

(3) 加减运算：包括补码加减、原码加减和移码加减运算。补码加减运算用于带符号整数，符号位和数值位一起运算。同号相加时，若结果的符号不同于加数的符号，则会发生溢出；因为IEEE 754标准用原码小数表示尾数，所以原码加减运算用于浮点数的尾数，符号位和数值位分开运算，同号数相加或异号数相减时，做加法，同号数相减或异号数相加时，做减法；因为IEEE 754标准用移码表示指数，所以移码加减运算用于浮点数的指数，移码的和、差等于和、差的补码，因此，可通过移码先求出和、差的补码，最后将符号取反，就能得到和、差的移码表示。减法运算电路只要在加法器基础上增加求补和溢出判断电路，并将进位输入端用于加减控制就可实现，因此，所有的减法运算都是用加法器实现的。

(4) 乘法运算：包括无符号数乘法、补码乘法和原码乘法，都可用加减及右移运算实现。无符号数乘法用于无符号整数；补码乘法用于带符号整数，符号位和数值位一起运算，可采用Booth算法或MBA算法；原码乘法用于浮点数尾数，符号位和数值位分开运算，数值部分用无符号数乘法实现。除了可以在定点运算部件中用加法和右移来实现乘法运算以外，也可用基于CSA的阵列乘法器、流水线乘法器、MBA＋WT乘法器等实现。两个n位数相乘得到$2n$位数乘积，若结果只取低n位，则乘积高n位必须是0(无符号数乘法)或符号(补码乘法)，否则溢出。对于一位乘法，n位数相乘大约需要n次加减运算和n次右移运算。

(5) 除法运算：包括无符号数除法、补码除法和原码除法，都可用加减及左移运算实现。无符号数除法用于无符号整数，有恢复余数法和不恢复余数法两种；补码除法用于带符号整数，符号位和数值位一起运算，也有恢复余数法和不恢复余数法两种；原码除法用于浮点数尾数，符号和数值分开运算，数值部分用无符号数除法实现。因为除法运算无法事先确定做加法还是减法，所以无法实现流水线除法器。两个n位数相除时，需要将被除数扩展成$2n$位数，对于不恢复余数法，n位数除法大约需要n次加减运算和n次左移运算。

4. 浮点运算及浮点运算器

计算机中大多用IEEE 754标准表示浮点数，因此，浮点运算主要针对IEEE 754标准浮点数。浮点运算由专门的浮点运算器实现，因为一个浮点数由一个定点小数和一个定点整数组成，所以浮点运算器由定点运算部件构成，其核心部件还是带先行进位加法器的ALU。浮点运算包括浮点加减运算和浮点乘除运算。

(1) 加减运算：按照对阶、尾数加减、规格化、舍入和溢出判断步骤完成。对阶时小阶向大阶看齐，阶小的那个数的尾数右移，直到两数阶码相同，右移时一般保留两位或三位附加位；尾数加减时用原码加减运算实现；规格化处理时根据结果的尾数形式的不同确定进行左规或右规操作；舍入操作有就近舍入、正向舍入、负向舍入和截去4种方式，默认的是就近

舍入到偶数方式;溢出判断主要根据结果的阶码进行判断,当发生阶码上溢时,运算结果发生溢出,当发生阶码下溢时,运算结果近似为0。

(2) 乘除运算:尾数用原码小数的乘/除运算实现,阶码用移码加减运算实现,需要对结果进行规格化、舍入和溢出判断。

3.3 基本术语解释

逻辑移位(Logical Shift)

逻辑移位是对无符号数进行的移位,把无符号数看成一个逻辑数进行移位操作。左移时,高位移出,低位补0;右移时,低位移出,高位补0。

算术移位(Arithmetic Shift)

算术移位是对带符号整数进行的,移位前后符号位不变。移位时,符号位不动,只是数值部分进行移位。左移时,高位移出,末位补0,移出非符时,发生溢出。右移时高位补符,低位移出。移出时进行舍入操作。

循环(逻辑)移位(Rotating Shift)

循环移位是一种逻辑移位,移位时把高(低)位移出的一位送到低(高)位,即左移时,各位左移一位,并把最左边的位移到最右边;右移时,各位右移一位,并把最右边的位移到最左边。

扩展操作(Extending)

在计算机内部,有时需要将一个取来的短数扩展为一个长数,此时要进行填充(扩展)处理。有“零扩展”和“符号扩展”两种。

零扩展(Zero Extending)

对无符号整数进行高位补0的操作称为“零扩展”。

符号扩展(Sign Extending)

对补码整数在高位直接补符的操作,称为“符号扩展”。

扩展器(Extender)

进行扩展(填充)操作的部件,称为扩展器。一般输入为n位,输出为$2n$位。

半加器(Half Adder)

只考虑本位两个加数而不考虑低位进位来生成本位和的一位加法器。

全加器(Full Adder,(3, 2) Adder)

不仅考虑本位两个加数而且考虑低位进位来生成本位和的一位加法器。

加法器(Adder)

能进行n位加法运算的部件。

行波进位(Ripple Carry)

在进行n位加法运算时,低位向高位的进位采用像“行波”一样串行传递的方式。

行波进位加法器(Ripple Carry Adder)

行波进位加法器也称为串行进位加法器,它通过n个全加器按照串行方式连接起来实现。进位方式采用行波进位方式。

先行(超前)进位(Carry LookAhead,CLA)

通过引入进位生成和进位传递两个进位辅助函数,使得加法器的各个进位之间相互独立、并行产生。这种进位方式也称为并行进位方式。

成组先行进位(Block Carry LookAhead,BCLA)

将数据分成若干组,在每一组内,除了并行生成组内各位向前面的进位外,同时还通过组进位生成和组进位传递这两个组进位辅助函数,促使各组的进位也能相互独立、并行产生。

先行进位加法器(CLA Adder)

采用先行进位方式实现的加法器,也称为并行进位加法器。因为采用先行进位方式能够快速得到和数,故也称为快速加法器。

算术逻辑部件(Arithmetic Logic Unit)

用于执行各种基本算术运算和逻辑运算的部件,其核心部件是加法器,有两个操作数输入端和低位进位输入端,一个运算结果输出端和若干标志信息(如零标志、溢出标志等)输出端。因为 ALU 能进行多种运算,因此,需要通过相应的操作控制输入端来选择进行何种运算。

零标志 ZF,溢出标志 OF,进位/借位标志 CF,符号标志 SF

ALU 部件的输出除了运算结果外,还有一组状态标志信息。例如,ZF(Zero Flag)为 1 时表示结果为 0;OF(Overflow Flag)为 1 时表示结果溢出;CF(Carry Flag)为 1 表示在最高位产生了进位或借位;SF(Sign Flag)和符号位保持一致,若为 1 则表示结果为负数。

布斯算法(Booth's Algorithm)

是一种一位补码乘法算法,用于带符号数的乘法运算,由 Booth 提出。算法的基本思想是在乘数的末位添加一个“0”,乘数中出现的连续“0”和连续“1”处不进行任何运算;出现“10”时,做减法;出现“01”时,做加法。每次只做一位乘法,因而每一步都右移一位。

改进布斯算法,基-4 Booth 算法(Modified Booth's Algorithm,MBA)

从布斯算法推导得到,采用两位一乘,根据乘数中连续三位的不同取值确定每一步相应的运算,每次部分积右移两位。

对阶(Align Exponent)

浮点数加/减运算时,在尾数相加/减之前所进行的操作称为对阶。对阶时,需要比较两个阶的大小。阶小的那个数的尾数右移,阶码增量。右移一次,阶码加 1,直到两数的阶码相等为止。

溢出(Overflow)

溢出是指一个数比给定的格式所能表示的最大值还要大,或比最小值还要小的现象。因为无符号数、带符号整数和浮点数的位数是有限的,所以,都有可能发生溢出,但判断溢出的具体方法不同。

阶码下溢(Exponent Underflow)

在浮点数运算中,当运算的结果其指数(阶码)比最小允许值还小,此时,运算结果发生阶码下溢,也即运算结果的实际值位于 0 和绝对值最小的可表示数之间。通常机器会把阶码下溢时浮点数的值置为 0。因此,这种情况下结果并没有发生错误,只是得到了一个近似于 0 的值,因而无须进行溢出处理。

阶码上溢(Exponent Overflow)

在浮点数运算中，当运算的结果其指数(阶码)超过了最大允许值，此时，浮点数发生了上溢。即向∞方向溢出。如果结果是正数，则发生正上溢，有的机器把值置为+∞；如果是负数，则发生负上溢，有的机器把值置为−∞。这种情况为软件故障，通常要引入溢出故障处理程序来处理。

规格化数(Normalized Number)

为了使浮点数中能尽量多地表示有效位数，一般要求运算结果用规格化数形式表示。规格化浮点数的尾数小数点后的第一位一定是个非零数。因此，对于原码编码的尾数来说，只要看尾数的第一位是否为1就行；对于补码表示的尾数，只要看符号位和尾数最高位是否相反。

左规(Left Normalize)

在浮点数运算中，当一个尾数的数值部分的高位出现0时，尾数为非规格化形式。此时，进行“左规”操作：尾数左移一位，阶码减1，直到尾数为规格化形式为止。

右规(Right Normalize)

在浮点数运算中，当尾数最高有效位有进位时，发生尾数溢出。此时，进行“右规”操作：尾数右移一位，阶码加1，直到尾数为规格化形式为止。右规过程中，要判断是否发生溢出。此时，只要阶码不发生上溢，那么浮点数就不会溢出。

舍入(Rounding)

舍入是指数值数据右部的低位数据需要丢弃时，为保证丢弃后数值误差尽量小而考虑的一种操作。例如，定点整数“右移”时、浮点加/减运算中某数“对阶”时、浮点运算结果“右规”时都会涉及舍入。

保护位(Guard Bit)和**舍入位**(Rounding Bit)

为了使浮点数的有效数据位在右移时最大限度地保证不丢失，一般在运算过程中得到的中间值后面增加若干数据位，这些位用来保存右移后的有效数据，因此，它们是添加的附加位。增设附加位后，能保证运算结果具有一定的精度，但最终必须将附加位去掉，以得到规定格式的浮点数，此时要考虑舍入。在IEEE 754标准中规定，浮点运算的中间结果可以额外多保留两位附加位，这两位分别称为保护位和舍入位。

粘位(Sticky Bit)

IEEE 754中规定，为了更进一步提高计算精度，可以在舍入位右边再增加一位，称为“粘位”，只要舍入位的右边还有任何非零数位，则粘位为1，否则为0。

运算器(Operational Unit)

即运算部件，通常指用ALU以及为了完成ALU的运算而必须与之共同工作的各种寄存器、多路选择器和实现数据传送的总线等构成的部件。根据功能不同有定点运算器和浮点运算器，它们是数据通路中的核心部件。

通用寄存器组(General Register Set,GRS)

CPU中提供了若干个通用寄存器，这些寄存器可以用来存放指令操作的对象，需要在指令中明确给出寄存器的编号，所有通用寄存器合起来构成一个通用寄存器组，也称为寄存器堆或寄存器文件(Register File)。通常，通用寄存器组有两个读口和一个写口。

多路选择器(Multiplexer)

在多个输入数据中根据控制信号选择其一作为输出的部件。

桶形移位器(Barrel Shifter)

由大量多路选择器实现的快速移位器,可一次左移或右移多位,移动位数由控制输入端给出。

Q 乘商寄存器(Q Multiplier-quotient Register)

Q 乘商寄存器用于乘除运算。在乘法中该寄存器用来存放乘数,在除法中用来存放商,并可以和另外的移位器共同完成左移(用于除法)或右移(用于乘法)操作。

3.4 常见问题解答

1. 无符号加法器如何实现?

答:计算机中,最基本的加法器是无符号加法器。根据进位方式的不同,有 3 种基本实现方式。串行进位加法器(行波进位加法器):通过 n 个全加器按照串行方式连起来实现。并行进位加法器(先行进位加法器):通过引入进位生成函数和进位传递函数,使得进位之间相互独立、并行产生。并行进位加法器也称为快速加法器。进位选择加法器:通过选择两个分别带进位 0 和 1 的高位部分加法器的输出来实现高、低两部分的并行执行。

2. 补码加法器如何实现?

答:两个 n 位补码进行加法运算的规则是:两个 n 位补码直接相加,并将结果中最高位的进位丢掉。也即采用模运算方式。显然,可用一个 n 位无符号加法器来生成各位的和。最终的结果是否正确,取决于结果是否溢出,只要不溢出,则结果一定是正确的。因此,补码加法器只要在无符号加法器的基础上再增加"溢出判断电路"即可。

3. 在补码加法器中,如何实现补码减法运算?

答:补码减法的规则是:两个数差的补码可用第一个数的补码加上另一数的负数的补码得到。由此可见,减法运算可在加法器中运行。只要在加法器的第二个输入端输入减数的负数的补码。求一个数的负数的补码电路称为"负数求补电路"。可以通过"各位取反、末尾加 1"来实现"负数求补电路"。

4. 现代计算机中是否要考虑原码加/减运算?

答:现代计算机中浮点数采用 IEEE 754 标准表示,因此在进行两个浮点数加减运算时,必须考虑原码的加减运算,因为,IEEE 754 规定浮点数的尾数都用原码表示。

5. 加法器的运算速度取决于什么?

答:在门电路延迟一定的情况下,加法器的速度主要取决于进位方式,先行进位方式比串行进位方式的速度快。

6. 定点整数运算要考虑增加保护位和舍入吗?

答:不需要。整数运算的结果还是整数,没有误差,无须考虑增加保护位,也无须考虑舍入。但运算结果可能会"溢出"。

7. 如何判断带符号整数运算结果是否溢出?

答:带符号整数用补码表示,对于单符号补码(即 2-补码)和双符号补码(即 4-补码,变形补码),其溢出判断方式不同。变形补码运算的溢出判断规则为:"当结果的两个符号位不同时,发生溢出"。单符号补码运算时,异号数相加不会溢出,而对于同号数相加,则有两种判断规则。规则 1 为:"若结果的符号与两个加数的符号不同,则发生溢出"。规则 2 为:

"若最高位的进位和次高位的进位不同,则发生溢出"。

8. 在计算机中,乘法和除法运算如何实现?

答:乘法和除法运算是通过加、减运算和左、右移位运算来实现的。只要用加法器和移位寄存器在控制逻辑的控制下就可以实现乘除运算。也可用专门的乘法器和除法器实现。

9. 浮点数如何进行舍入?

答:舍入方法选择的原则是:(1)尽量使误差范围对称,使得平均误差为0,即有舍有入,以防误差积累。(2)方法要简单,以加快速度。

IEEE 754有4种舍入方式:(1)就近舍入:舍入为最近可表示的数,若结果值正好落在两个可表示数的中间,则一般选择舍入结果为偶数。(2)正向舍入:朝$+\infty$方向舍入,即取右边的那个数。(3)负向舍入:朝$-\infty$方向舍入,即取左边的那个数。(4)截去:朝0方向舍入。即取绝对值较小的那个数。

10. 在C语言程序中,为什么以下程序段最终的f值为0,而不是2.5?

```
float f=2.5+1e10;
f=f-1e10;
```

答:首先,float类型采用IEEE 754单精度浮点数格式表示,因此,最多有24位二进制有效位数。因为$1e10=10^{10}=10\times10^{3}\times10^{6}$,在数量级上大约相当于$2^{3}\times2^{10}\times2^{20}=2^{33}$,而2.5的数量级为$2^{1}$,因此,在计算2.5+1e10进行对阶时,两数阶码的差为32,也就是说,2.5的尾数要向右移32位,从而使得24位有效数字全部丢失,尾数变为全0,再与1e10的尾数相加时结果就是1e10的尾数,因此$f=2.5+1e10$的运算结果仍为1e10,这样,再执行$f=f-1e10$时结果就为0。这就是典型的大数吃小数的例子。

3.5 单项选择题

1. 8位无符号整数1001 0101B右移一位后的值为(　　)。
   A. 0100 1010B　　B. 0100 1011B
   C. 1000 1010B　　D. 1100 101B
2. 8位补码定点整数1001 0101B右移一位后的值为(　　)。
   A. 0100 1010B　　B. 0100 1011B
   C. 1000 1010B　　D. 1100 1010B
3. 8位补码定点整数1001 0101B左移一位后的值为(　　)。
   A. 1010 1010B　　B. 0010 1010B
   C. 0010 1011B　　D. 溢出
4. 8位补码定点整数1001 0101B扩展8位后的值用十六进制表示为(　　)。
   A. 0095H　　B. 9500H
   C. FF95H　　D. 95FFH
5. 原码定点小数1.1001 0101B扩展8位后的值为(　　)。
   A. 1.0000 0000 1001 0101B　　B. 1.1001 0101 0000 0000B
   C. 1.1111 1111 1001 0101B　　D. 1.1001 0101 1111 1111B

6. 考虑以下C语言代码：

```
short si=-8196;
int i=si;
```

执行上述程序段后，i的机器数表示为(　　)。

A. 0000 9FFCH　　B. 0000 DFFCH

C. FFFF 9FFCH　　D. FFFF DFFCH

7. CPU中能进行算术和逻辑运算的最基本运算部件是(　　)。

A. 多路选择器　　B. 移位器

C. 加法器　　D. ALU

8. ALU的核心部件是(　　)。

A. 多路选择器　　B. 移位器

C. 加法器　　D. 寄存器

9. 假定T表示一级门延迟，一个异或门的延迟为3T，不考虑线延迟，则8位全先行进位加法器的关键路径延迟为(　　)。

A. 6T　　B. 8T　　C. 16T　　D. 17T

10. 74181 ALU的功能是(　　)。

A. 实现16种4位算术运算

B. 实现16种4位逻辑运算

C. 实现16种4位算术和逻辑运算

D. 实现4位乘法运算和4位除法运算

11. 在补码加/减运算部件中，无论采用双符号位还是单符号位，必须有(　　)电路，它一般用异或门来实现。

A. 译码　　B. 编码　　C. 溢出判断　　D. 移位

12. 某计算机字长为8位，其CPU中有一个8位加法器。已知无符号数 $x=69$，$y=38$，现要在该加法器中完成 $x+y$ 的运算，此时该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 0100 0101B、0010 0110B、0　　B. 0100 0101B、0010 0110B、1

C. 0100 0101B、1101 1010B、0　　D. 0100 0101B、1101 1010B、1

13. 某计算机字长为8位，其CPU中有一个8位加法器。已知无符号数 $x=69$，$y=38$，现要在该加法器中完成 $x-y$ 的运算，此时该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 0100 0101B、0010 0110B、0　　B. 0100 0101B、1101 1001B、1

C. 0100 0101B、1101 1010B、0　　D. 0100 0101B、1101 1010B、1

14. 某计算机字长为8位，其CPU中有一个8位加法器。已知带符号整数 $x=-69$，$y=-38$，现要在该加法器中完成 $x+y$ 的运算，此时该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 1011 1011B、1101 1010B、0　　B. 1011 1011B、1101 1010B、1

C. 1011 1011B、0010 0101B、0　　D. 1011 1011B、0010 0101B、1

15. 某计算机字长为8位，其CPU中有一个8位加法器。已知带符号整数 $x=-69$，$y=-38$，现要在该加法器中完成 $x-y$ 的运算，此时该加法器的两个输入端信息和输入的低位进位信息分别为(　　)。

A. 1011 1011B、1101 1010B、0　　B. 1011 1011B、1101 1010B、1

C. 1011 1011B、0010 0110B、0　　D. 1011 1011B、0010 0101B、1

16. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$x=63$，$y=-31$，则 $x+y$ 的机器数及其相应的溢出标志OF分别是(　　)。

A. 1FH、0　　B. 20H、0

C. 1FH、1　　D. 20H、1

17. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$x=63$，$y=-31$，则 $x-y$ 的机器数及其相应的溢出标志OF分别是(　　)。

A. 5DH、0　　B. 5EH、0

C. 5DH、1　　D. 5EH、1

18. 某8位计算机中，假定带符号整数变量 x 和 y 的机器数用补码表示，$[x]_{补}=F5H$，$[y]_{补}=7EH$，则 $x+y$ 的值及其相应的溢出标志OF分别是(　　)。

A. 115、0　　B. 119、0

C. 115、1　　D. 119、1

19. 某8位计算机中，假定带符号整数变量 x 和 y 的机器数用补码表示，$[x]_{补}=F5H$，$[y]_{补}=7EH$，则 $x-y$ 的值及其相应的溢出标志OF分别是(　　)。

A. 115、0　　B. 119、0

C. 115、1　　D. 119、1

20. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x+2y$ 的机器数以及相应的溢出标志OF分别是(　　)。

A. 32H、0　　B. 32H、1

C. FCH、0　　D. FCH、1

21. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x-2y$ 的机器数以及相应的溢出标志OF分别是(　　)。

A. 68H、0　　B. 68H、1

C. 8CH、0　　D. 8CH、1

22. 某8位计算机中，假定 x 和 y 是两个带符号整数变量，用补码表示，$[x]_{补}=44H$，$[y]_{补}=DCH$，则 $x/2+2y$ 的机器数以及相应的溢出标志OF分别是(　　)。

A. CAH、0　　B. CAH、1

C. DAH、0　　D. DAH、1

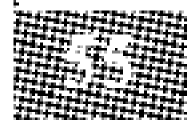
23. 假定有两个整数用8位补码分别表示为 $r1=F5H$，$r2=EEH$。若将运算结果存放在一个8位寄存器中，则下列运算中会发生溢出的是(　　)。

A. $r1+r2$　　B. $r1-r2$

C. $r1\times r2$　　D. $r1/r2$

24. 以下关于原码一位乘法算法要点的描述中，错误的是(　　)。

A. 符号位和数值位分开运算，符号位可由一个异或门生成



B. 通过循环执行“加法”和“移位”操作得到乘积

C. ALU中是否进行部分积与被乘数的加法运算由乘数最低位决定

D. 移位时，将进位位、部分积和乘积部分一起进行算术右移

25. 假定一次ALU运算用1个时钟周期，移位一次用1个时钟周期，则最快的32位原码一位乘法所需要的时钟周期数大约为(　　)。

A. 32　　B. 64　　C. 96　　D. 100

26. 以下关于Booth补码一位乘法算法要点的描述中，错误的是(　　)。

A. 符号位和数值位一起参加运算，无须专门的符号生成部件

B. 通过循环执行“加/减”和“移位”操作得到乘积

C. 由乘数最低两位决定对部分积和被乘数进行何种运算

D. 移位时，将进位位、部分积和乘积部分一起进行算术右移

27. 以下关于乘法运算部件的叙述中，错误的是(　　)。

A. 补码乘法部件可用于带符号整数的乘法运算

B. 原码乘法部件可用于浮点数中尾数相乘运算

C. 快速阵列乘法器中的基本部件包含移位器

D. 两位乘法运算比一位乘法运算速度约快一倍

28. 对于两个n位无符号整数除法运算，以下关于不恢复余数算法要点的描述中，错误的是(　　)。

A. 起始时被除数在高位扩展n位0，以将其扩展为$2n$位无符号整数

B. 为判断中间余数的正/负，需在余数寄存器的最高位前增加一位符号位

C. 至少需$n+1$次循环执行“加/减”和“左移”操作才能得到n位商

D. 运算结果一定不会发生溢出，故无须通过得到最高位商来判断溢出

29. 对于IEEE 754单精度浮点数加减运算，在对阶过程中，需要计算两个阶码Ex和Ey之差的补码$[\Delta E]_{补}$。若$\Delta E \geqslant 128$或$\Delta E \leqslant -129$，则$[\Delta E]_{补}$发生溢出。假定$[Ex]_{移}$、$[-[Ey]_{移}]_{补}$和$[\Delta E]_{补}$的最高有效位分别记为Exs、Eys和Ebs，则相应的溢出判断方程为(　　)。

A. $Overflow=\overline{Exs}\ \overline{Eys}\, Ebs+Exs\, Eys\, \overline{Ebs}$

B. $Overflow=\overline{Exs}\, Eys\, \overline{Ebs}+Exs\, \overline{Eys}\ \overline{Ebs}$

C. $Overflow=\overline{Exs}\, Eys\, Ebs+Exs\, \overline{Eys}\, Ebs$

D. $Overflow=\overline{Exs}\ \overline{Eys}\ \overline{Ebs}+Exs\, Eys\, Ebs$

30. IEEE 754单精度浮点数加减运算的对阶过程中，需要计算两个阶码Ex和Ey之差的补码$[\Delta E]_{补}$。假设两个浮点数分别记为$[X]_{浮}$和$[Y]_{浮}$，$[Ex]_{移}$、$[Ey]_{移}$和$[\Delta E]_{补}$的最高有效位分别记为Exs、Eys和Ebs，当$[\Delta E]_{补}$发生溢出时，正确的处理方式是(　　)。

A. 中止当前程序的执行，调出相应的“溢出”异常处理程序执行

B. 当Exs为1时置最终结果为$[X]_{浮}$；当Exs为0时置最终结果为$[Y]_{浮}$

C. 当Eys为1时置最终结果为$[X]_{浮}$；当Eys为0时置最终结果为$[Y]_{浮}$

D. 当Ebs为0时置最终结果为$[X]_{浮}$；当Ebs为1时置最终结果为$[Y]_{浮}$

31. 若两个(用 IEEE 754 单精度浮点格式表示)的 float 型变量 x 和 y 的机器数分别表示为 x=40E8 0000H,y=C204 0000H,则在计算 $x+y$ 时,第一步对阶操作的结果$[\Delta E]_补$为(　　)。

A. 0000 0111B　　B. 0000 0011B

C. 1111 1011B　　D. 1111 1101B

32. 对于 IEEE 754 单精度浮点数加减运算,只要对阶时得到的两个阶码之差的绝对值$|\Delta E|$大于等于(　　),就无须继续进行后续处理,此时,运算结果直接取阶大的那个数。

A. 23　　B. 24　　C. 126　　D. 128

33. IEEE 754 标准提供了以下 4 种舍入模式,其中平均误差最小的是(　　)。

A. 就近舍入(中间值时强迫为偶数)

B. 正向舍入(即朝+∞方向舍入)

C. 负向舍入(即朝-∞方向舍入)

D. 截断舍入(即朝 0 方向舍入)

【参考答案】

1. A	2. D	3. D	4. C	5. B	6. D	7. D
8. C	9. A	10. C	11. C	12. A	13. B	14. A
15. D	16. B	17. B	18. A	19. D	20. C	21. D
22. C	23. C	24. D	25. B	26. D	27. C	28. C
29. D	30. B	31. D	32. B	33. A		

3.6　分析应用题

1. 考虑下列 C 语言程序代码:

```
int i=65535;
short si=(short)i;
int j=si;
```

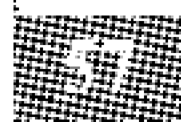
假定上述程序段在某 32 位机器上执行,sizeof (int)=4,则变量 i、si 和 j 的值分别是多少? 为什么?

【分析解答】

在一台 32 位机器上执行上述代码段时,i 为 32 位补码表示的定点整数,第 2 行要求强行将一个 32 位带符号数截断为 16 位带符号整数,65535 的 32 位补码表示为 0000 FFFFH,截断为 16 位后变成 FFFFH,它是-1 的 16 位补码表示,因此 si 的值是-1。再将该 16 位带符号整数扩展为 32 位时,就变成了 FFFF FFFFH,它是-1 的 32 位补码表示,因此 j 的值也为-1。也就是说,i 的值原来为 65535,经过截断、再扩展后,其值变成了-1。

2. 考虑以下 C 语言程序代码:

```
int   func1 (unsigned word)
{
    return  (int) ((word<<24)>>24);
```



```
}
int   func2 (unsigned word)
{
    return  ((int) word<<24)>>24;
}
```

假设在一个 32 位机器上执行这些函数，sizeof(int)=4。说明函数 func1 和 func2 的功能，并填写表 3.1，给出对表中“异常”数据的说明。

表 3.1　题 2 用表

W		func1(*w*)		func2(*w*)	
机器数	值	机器数	值	机器数	值
	127				
	128				
	255				
	256				

【分析解答】

函数 func1 的功能是把无符号数高 24 位清零(左移 24 位再逻辑右移 24 位)，结果一定是正的带符号整数；而函数 func2 的功能是把无符号数的高 24 位都变成和第 25 位一样，因为左移 24 位后原来的第 25 位变为左边第一位，然后进行算术右移，高位补符号，即高 24 位都变成和原来第 25 位相同。程序执行的结果如表 3.2 所示，表中机器数用十六进制表示。

表 3.2　题 2 填入结果后的表

W		func1(*w*)		func2(*w*)	
机器数	值	机器数	值	机器数	值
0000007FH	127	0000007FH	+127	0000007FH	+127
00000080H	128	00000080H	+128	**FFFFFF80H**	**−128**
000000FFH	255	000000FFH	+255	**FFFFFFFFH**	**−1**
00000100H	256	**00000000H**	**0**	**00000000H**	**0**

上述表 3.2 中，加粗数据是一些“异常”结果。当 $w=128$ 和 $w=255$ 时，第 25 位正好是 1，因此函数 func2 执行的结果为一个负数，出现了“异常”。当 $w=256$ 时，低 8 位为 00，高 24 位为非 0 值，左移 24 位后使得有效数字被移出，因而发生了“溢出”，导致出现了“异常”结果 0。

3. 以下是两段 C 语言代码，函数 arith 是直接用 C 语言写的，而 optarith 是对 arith 函数以某个确定的 M 和 N 编译生成的机器代码进行反编译而成的。根据 optarith 推断函数 arith 中 M 和 N 的值各是多少？

```
#define    M
```

```
#define    N
int arith (int x, int y)
{
    int  result=0;
    result=x*M+y/N;
    return result;
}

int optarith (int x, int y)
{
    int  t=x;
    x <<=4;
    x-=t;
    if (y<0)  y+=3;
    y>>=2;
    return x+y;
}
```

【分析解答】

对反编译结果进行分析，可知：对于 x，指令机器代码中有一条"x 左移 4 位"指令，即 $x \leftarrow 16x$，然后有一条"减法"指令，即 $x \leftarrow 16x - x = 15x$，根据源程序知 $M=15$；对于 y，有一条"y 右移 2 位"指令，即 $y \leftarrow y/4$，根据源程序知 $N=4$。但是，当 $y<0$ 时，对于有些 y，执行 $y>>2$ 后的值并不等于 $y/4$。例如，当 $y=-1$ 时，在反编译函数 optarith 中执行 $y>>2$ 时，因为 -1 的机器数为全 1，右移两位后还是全 1，即 $-1>>2=-1$，结果为 -1；而原函数 arith 中执行 $y/4$ 时，因为 $-1/4=0$，得到结果为 0。同样，$y=-2$、$y=-3$ 和 $y=-5$ 等情况下都会发生 $y>>2 \neq y/4$ 的结果。因为当 $y<0$ 时，$(y+3)/4=y/4$，所以，函数 optarith 中在执行 $y>>2$ 之前加了一条语句"if (y<0) y+=3;"，以对 y 进行调整。

4. 设 $A_4 \sim A_1$ 和 $B_4 \sim B_1$ 分别是 4 位加法器的两组输入，C_0 为低位来的进位。当加法器分别采用串行进位和先行进位时，写出 4 个进位 $C_4 \sim C_1$ 的逻辑表达式。

【分析解答】

串行进位：

$$C_1 = A_1B_1 + A_1C_0 + B_1C_0; \quad C_2 = A_2B_2 + A_2C_1 + B_2C_1;$$
$$C_3 = A_3B_3 + A_3C_2 + B_3C_2; \quad C_4 = A_4B_4 + A_4C_3 + B_4C_3;$$

并行进位：

$$C_1 = A_1B_1 + (A_1 + B_1)C_0$$
$$C_2 = A_2B_2 + (A_2 + B_2)A_1B_1 + (A_2 + B_2)(A_1 + B_1)C_0$$
$$C_3 = A_3B_3 + (A_3 + B_3)A_2B_2 + (A_3 + B_3)(A_2 + B_2)\ A_1B_1 + (A_3 + B_3)(A_2 + B_2)(A_1 + B_1)C_0$$
$$C_4 = A_4B_4 + (A_4 + B_4)A_3B_3 + (A_4 + B_4)(A_3 + B_3)A_2B_2 + (A_4 + B_4)(A_3 + B_3)(A_2 + B_2)A_1B_1 + (A_4 + B_4)(A_3 + B_3)(A_2 + B_2)(A_1 + B_1)C_0$$

5. 说明如何用 SN74181 和 SN74182 器件设计一个 16 位先行进位补码加/减运算部

件，并给出零标志、进/借位标志、溢出标志、符号标志的逻辑表达式。

【分析解答】

假定两个操作数分别为 $A=A_{15}\sim A_0$，$B=B_{15}\sim B_0$，结果用 $F_{15}\sim F_0$ 表示，最高位向前的进位用 C_{16} 表示。首先用 4 个 SN74181 和 1 个 SN74182 构成一个 16 位无符号数先行进位加法器，然后在 4 个 SN74181 的输入端 B 的每位加一个异或门，异或门的一个输入是 B_i，另一个输入是最低位进位 C_0，即 $B_i'=B_i\oplus C_0$，这样在 $C_0=1$ 时作减法，当 $C_0=0$ 时作加法。(详细设计可参考主教材[①]中图 3.15 和图 3.16)

零标志 ZF 的逻辑表达式为 $ZF=\overline{F_{15}+\cdots+F_0}$；通常，无符号数加、减运算和带符号整数(补码)加、减运算都在上面描述的同一个加/减运算部件中执行，进/借位标志 CF 在带符号整数(补码)运算中没有意义，因此只需根据无符号数运算时 CF 的含义来考虑 CF 的逻辑表达式。对于无符号加/减运算，当 $C_0=0$ 时，$CF=C_{16}$，而当 $C_0=1$ 时，$CF=\overline{C_{16}}$，所以 $CF=C_0\oplus C_{16}$；溢出标志 OF 对无符号数加/减运算没有意义，因此，只需根据补码加/减运算中 OF 的含义来考虑 OF 的逻辑表达式。在补码加/减运算中，若两个加数的符号相同，但不同于结果的符号，则结果肯定溢出，因此，$OF=\overline{A_{15}}\ \overline{B_{15}'}\ F_{15}+A_{15}\ B_{15}'\ \overline{F_{15}}$；符号标志 SF 只对补码加/减运算有意义，当结果为负数时，$F_{15}=1$，故 $SF=F_{15}$。

6. 说明如何用 SN74181 和 SN74182 器件设计一个 32 位的 ALU，要求采用两级先行进位结构，给出所需要的 SN74181 和 SN74182 芯片数。

【分析解答】

首先用 4 个 SN74181 和 1 个 SN74182 构成一个 16 位无符号数先行进位加法器，然后在输入端 B 增加“取补”电路以构成补码加/减运算部件，这样，得到一个 16 位两级先行进位加/减运算部件。将这样的两个 16 位加/减运算部件串联起来，串联时，注意将低 16 位对应的那个部件的最高位进位 C_{16} 作为高 16 位对应的那个部件的最低位进位，并且，高 16 位对应的那个部件中，在 SN74181 的输入端 B 处的异或门的其中一个输入为 B_i，另一个输入则与低 16 位对应部件的相应异或门输入端一样，都是 C_0，而不是 C_{16}。一共需要 8 个 SN74181 和 2 个 SN74182。

7. 某字长为 8 位的计算机中，x 和 y 为无符号整数，已知 $x=68$，$y=80$，x 和 y 分别存放在寄存器 A 和 B 中。请回答下列问题(要求最终用十六进制表示二进制序列)。

(1) 寄存器 A 和 B 中的内容分别是什么？

(2) 若 x 和 y 相加后的结果存放在寄存器 C 中，则寄存器 C 中的内容是什么？运算结果是否正确？此时，零标志 ZF 是什么？加法器最高位的进位 C_n 是什么？

(3) 若 x 和 y 相减后的结果存放在寄存器 D 中，则寄存器 D 中的内容是什么？运算结果是否正确？此时，零标志 ZF 是什么？加法器最高位的进位 C_n 是什么？

(4) 无符号整数加/减运算时，加法器最高位进位 C_n 的含义是什么？它与进/借位标志 CF 的关系是什么？

(5) 无符号整数一般用来表示什么信息？为什么通常不对无符号整数的运算结果判断溢出？

---

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)

【分析解答】

(1) $x=68=01000100B=44H$；$y=80=01010000B=50H$。所以，寄存器 A 和 B 中的内容分别是 44H 和 50H。

(2) $x+y=01000100B+01010000B=(0)\ 10010100B=94H$，所以，寄存器 C 中的内容为 94H，对应的真值为 148，运算结果正确。因为结果不为 0，所以 $ZF=0$；加法器最高位的进位 C_n 为 0。

(3) $x-y=x+[-y]_{补}=01000100B+10110000B=(0)\ 11110100B=F4H$，所以，寄存器 D 中的内容为 F4H，对应的真值为 244，运算结果不正确，这是因为相减结果为负数造成的。因为结果不为 0，所以 $ZF=0$；加法器最高位的进位 C_n 为 0。

(4) 在加法器中进行无符号整数加法运算时，若加法器最高位进位 $C_n=1$，则表示实际结果大于最大可表示数 255；在加法器中进行无符号整数减法运算时，若加法器最高位进位 $C_n=1$，则表示被减数大于减数，反之被减数小于减数。因此，在无符号数相加时，CF 就等于 C_n，表示进位；在无符号数相减时，通常将最高进位 C_n 取反来作为借位标志 CF，即无符号整数相减时，$CF=\overline{C}_n$，$CF=1$ 表示有借位。

(5) 无符号整数一般用来表示地址(指针)信息，当两个地址相加结果大于最大地址而取低位地址时，相当于取模，即采用地址或指针的循环运算。因此通常不需要判断其运算结果是否溢出，即不考虑溢出标志 OF。

8. 假设某字长为 8 位的计算机中，带符号整数采用补码表示，$x=-68$，$y=-80$，x 和 y 分别存放在寄存器 A 和 B 中。请回答下列问题(要求最终用十六进制表示二进制序列)。

(1) 寄存器 A 和 B 中的内容分别是什么?

(2) 若 x 和 y 相加后的结果存放在寄存器 C 中，则寄存器 C 中的内容是什么? 运算结果是否正确? 此时，溢出标志 OF、符号标志 SF 和零标志 ZF 各是什么? 加法器最高位的进位 C_n 是什么?

(3) 若 x 和 y 相减后的结果存放在寄存器 D 中，则寄存器 D 中的内容是什么? 运算结果是否正确? 此时，溢出标志 OF、符号标志 SF 和零标志 ZF 各是什么? 加法器最高位的进位 C_n 是什么?

(4) 若将加法器最高位的进位 C_n 作为进位标志 CF，则能否直接根据 CF 的值对两个带符号整数的大小进行比较?

【分析解答】

(1) $[-68]_{补}=[-1000100B]_{补}=10111100B=BCH$。$[-80]_{补}=[-1010000B]_{补}=10110000B=B0H$。所以，寄存器 A 和 B 中的内容分别是 BCH 和 B0H。

(2) $[x+y]_{补}=[x]_{补}+[y]_{补}=10111100B+10110000B=(1)\ 01101100B=6CH$，最高位前面的一位 1 被丢弃，因此，寄存器 C 中的内容为 6CH，对应的真值为 +108，结果不正确。溢出标志位 OF 可采用以下任意一条规则判断得到。规则 1：若两个加数的符号位相同，但与结果的符号位相异，则溢出。规则 2：若最高位上的进位和次高位上的进位不同，则溢出。通过这两个规则都能判断出结果溢出，即溢出标志位 OF 为 1，说明寄存器 C 中的内容不是正确的结果。$x+y$ 的正确结果应是 $-68+(-80)=-148$，而运算的结果为 108，显然两者不等。其原因是 $x+y$ 的值(即 -148)小于 8 位补码可表示的最小值(即 -128)，也即结果发生了溢出；结果的第一位 0 为符号标志 SF，表示结果为正数。但因为溢出标志为

1,所以符号标志实际上也是错的;因为结果不为0,所以零标志 $ZF=0$;加法器最高位向前的进位 C_n 为1。

(3) $[x-y]_{补}=[x]_{补}+[-y]_{补}$ =10111100B+01010000B=(1) 00001100B=0CH,最高位前面的一位1被丢弃,因此,寄存器D中的内容为0CH,对应的真值为+12,结果正确。两个加数的符号位相异一定不会溢出,因此溢出标志 OF 为0,说明寄存器D中的内容是真正的结果;结果的第一位0为符号标志 SF,表示结果为正数;因为结果不为0,所以零标志 $ZF=0$;加法器最高位向前的进位 C_n 为1。

(4) 若将加法器最高位的进位 C_n 作为进位标志 CF,则无法直接根据 CF 的值判断两个带符号整数的大小,因此带符号加减运算中不考虑 CF 标志。

9. 某计算机标志寄存器包含4个标志位:CF-进/借位标志;OF-溢出标志;SF-符号标志;ZF-零标志。请说明在无符号数和带符号整数两种情况下,以下各种比较运算的逻辑判断表达式。

(1)等于;(2)大于;(3)小于;(4)大于等于;(5)小于等于。

【分析解答】

要比较两个数的大小,通常对这两个数先作减法,根据相减的结果生成相应的标志位,最后根据标志位判断大小。在无符号数相减时,一般不考虑 SF 和 OF 标志;在带符号整数相减时,一般不考虑 CF 标志。假设被减数的机器数为 X,减数的机器数为 Y,则在加法器中计算两数的差时,计算公式为 $X-Y=X+(-Y)_{补}$。以下举两个例子来说明。假定 $X=$1001B,$Y=$1100B,则在4位加法器中执行以下运算:1001B−1100B=1001B+0100B=(0)1101B。若是无符号数比较,则是9和12相比,显然,$ZF=0$,$CF=1$;若是带符号整数(补码表示),则是−7和−4比较,显然,$ZF=0$,$OF=0$,$SF=1$。假定X=1001B,Y=0100B,则在4位加法器中执行以下运算:1001B−0100B=1001B+1100B=(1)0101B。若是无符号数比较,则是9和4相比,显然,$ZF=0$,$CF=0$;若是带符号整数,则是−7和4比较,显然,$ZF=0$,$OF=1$,$SF=0$。

以下分别说明无符号数和带符号整数两种情况下各种比较运算的逻辑判断表达式。

a. 无符号数情况

(1) 等于:相减后结果为0,即 $F=ZF$。

(2) 大于:没有借位且相减后不为0,即 $F=\overline{CF+ZF}$。

(3) 小于:有借位且相减后不为0,即 $F=CF\cdot\overline{ZF}$。

(4) 大于等于:没有借位或相减后结果为0,即 $F=\overline{CF}+ZF$。

(5) 小于等于:有借位或相减后结果为0,即 $F=CF+ZF$。

b. 带符号整数情况

(1) 等于:相减后结果为0,即 $F=ZF$。

(2) 大于:结果不为0,且不溢出时为正,溢出时为负。即 $F=\overline{ZF}\cdot\overline{(SF\oplus OF)}$。

(3) 小于:结果不为0,且不溢出时为负,溢出时为正。即 $F=\overline{ZF}\cdot(SF\oplus OF)$。

(4) 大于等于:结果为0,或者,不溢出时为正,溢出时为负。即 $F=ZF+\overline{(SF\oplus OF)}$。

(5) 小于等于:结果为0,或者,不溢出时为负,溢出时为正。即 $F=ZF+(SF\oplus OF)$。

10. 填写表 3.3,注意对比无符号数和带符号整数的乘法结果,以及截断操作前、后的结果。

表 3.3 题 10 用表

模式	x		y		x×y(截断前)		x×y(截断后)	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110		010					
二进制补码	110		010					
无符号数	001		111					
二进制补码	001		111					
无符号数	111		111					
二进制补码	111		111					

【分析解答】

根据无符号数乘法运算和补码乘法运算算法,填写表 3.4。

表 3.4 题 10 填入结果后的表

模式	x		y		x×y(截断前)		x×y(截断后)	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110	6	010	2	001100	12	100	4
二进制补码	110	−2	010	+2	111100	−4	100	−4
无符号数	001	1	111	7	000111	7	111	7
二进制补码	001	+1	111	−1	111111	−1	111	−1
无符号数	111	7	111	7	110001	49	001	1
二进制补码	111	−1	111	−1	000001	+1	001	+1

对表 3.4 中结果分析如下:①两个数作为无符号数进行乘法运算和作为带符号整数进行乘法运算时,因为其所用的乘法算法不同,所以,乘积的机器数可能不同;②对于 n 位乘法运算,无论是无符号数乘法还是带符号整数乘法,若截取 $2n$ 位乘积的低 n 位作为最终的乘积,则都有可能结果溢出,即 n 位数字无法表示正确的乘积。表中给出的例子中,虽然带符号整数乘积截断后都没有发生溢出,但实际上还是存在溢出的情况,例如,011B×011B=001001B,截断后 011B×011B=001B,显然截断后的结果溢出。

11. 考虑以下 C 语言程序代码:

```
int  func1(unsigned short si)
{
   return (si* 256);
}
```

```
int  func2(unsigned short si)
{
    return (si/256);
}
int  func3(unsigned short si)
{
    return (((short) si* 256)/256);
}
int  func4(unsigned short si)
{
    return (short) ((si* 256)/256);
}
```

请回答下列问题：

(1) 假设计算机硬件不提供乘除运算功能，能否用移位运算实现上述函数功能？函数 func1、func2、func3 和 func4 得到的结果各有什么特征？

(2) 填写表 3.5(要求机器数用十六进制表示)，并对表中的"异常"数据进行分析。

表 3.5　题 11 用表

si		func1(*si*)		func2(*si*)		func3(*si*)		func4(*si*)	
机器数	值	机器数	值	机器数	值	机器数	值	机器数	值
	127								
	128								
	255								
	256								
	65535								

(3) 对于函数 func1 来说，用一位乘运算所花的时间开销大约是用移位运算的多少倍？对于函数 func2 来说，用不恢复余数除法所花的开销大约是用移位运算的多少倍？

【分析解答】

(1) 因为 $256=2^8$，所以上述函数中的乘、除运算可以分别用左、右移运算来实现。可用"左移 8 位"代替"乘 256"的操作，用"右移 8 位"代替"除以 256"的操作。func1(*si*)相当于将 *si* 逻辑左移 8 位，结果的最后 8 位都为 0；func2(*si*)相当于将 *si* 逻辑右移 8 位，结果的范围在 0～255 之间；func3(*si*)相当于将 *si* 先算术左移 8 位，再算术右移 8 位，所以结果的范围在－128 和 127 之间；func4(*si*)相当于将 *si* 先逻辑左移 8 位，再逻辑右移 8 位，最后以带符号整数类型返回。因为最后是逻辑右移，高位补 0，所以，返回的总是正数，结果的范围在 0 到 255 之间。

(2) 函数 func1、func2、func3 和 func4 的执行结果填表 3.6。

表 3.6　题 11 填入结果后的表

si		func1(si)		func2(si)		func3(si)		func4(si)	
机器数	值	机器数	值	机器数	值	机器数	值	机器数	值
007FH	127	7F00H	32512	0000H	0	007FH	127	007FH	127
0080H	128	8000H	32768	0000H	0	**FF80H**	**−128**	0080H	128
00FFH	255	FF00H	65280	0000H	0	**FFFFH**	**−1**	00FFH	255
0100H	256	**0000H**	**0**	0001H	1	**0000H**	**0**	**0000H**	**0**
FFFFH	65535	**FF00H**	**65280**	00FFH	255	**FFFFH**	**−1**	**00FFH**	**255**

上述表 3.6 中，加粗数据是一些“异常”结果。当 $si=256$ 时，由于 $256\times si=65536$，因此用 16 位无符号数无法表示实际结果，导致 func1(*si*)、func3(*si*)和 func4(*si*)都为 0；同样，当 $si=65535$ 时，由于 $256\times si$ 溢出，导致 func1、func3 和 func4 的函数值出现了“异常”结果；当 $si=128$ 和 $si=255$ 时，由于截断低 16 位乘积得到的第一位(符号位)为 1，因此再进行算术右移时高位补了 8 位 1，导致 func3 的函数值“溢出”，出现了“异常”结果。

(3) 用移位操作代替乘、除运算，其程序执行时间会大大减少。采用桶形移位器进行移位时，移动任何位数都只要一次移位操作。假定一次移位操作所用时间和一次加/减运算时间一样，都是一个时钟周期 T，那么，对于函数 func1，采用一位乘法器时，两个 16 位数相乘的运算所需循环次数为 16，每个循环内进行“判断-加法-右移”操作，其中判断操作是控制器在送出控制信号之前进行的，无须一个专门的时钟周期来实现，因此 16 位乘法运算所用时间大约为 $32T$，由此可知，采用一位乘法运算，函数 func1 的执行时间大约是采用移位运算的 32 倍。对于函数 func2，采用不恢复余数法时，两个 16 位数相除时所需循环次数为 16，循环内有“判断-左移(上商)-加/减”操作，所用时间大约为 $32T$，因此是采用移位运算的 32 倍左右。

12. 若一次加法需要 1ns，一次移位需要 0.5ns。请分别估算用一位乘法、两位乘法、基于 CRA 的阵列乘法、基于 CSA 的阵列乘法 4 种方式计算两个 8 位无符号二进制数乘积时所需要的时间。

【分析解答】

一位乘法需 8 次右移，8 次加法，共计 12ns；二位乘法需 4 次右移，4 次加法，共计 6ns；对于基于 CRA 的阵列乘法，每一级部分积不仅依赖于上一级部分积，还依赖于上一级最终的进位，而每一级进位又是串行进行的，所以最长的路径总共经过了 $8+2\times(8-1)=22$ 个单元节点(细胞模块)，假定经过每个单元节点所花时间平均为 0.5ns，则共计大约 11ns；对于基于 CSA 的阵列乘法，本级进位与本级和能同时传送到下一级，且同级部分积之间互不依赖，因此只需进行 O(N)次简单加法(即半加或全加)运算，假定简单加法时间为 8 位加法器时间的一半，则一共大约为 4ns。

13. 已知 $x=10$，$y=-6$，采用 6 位机器数表示。请按如下要求计算，并把结果还原成真值。

(1) 求$[x+y]_{补}$，$[x-y]_{补}$。

(2) 用原码一位乘法计算$[x\times y]_{原}$。

(3) 用补码一位乘法(Booth 算法)计算$[x\times y]_{补}$。

(4) 用 MBA(基-4 布斯)乘法计算$[x\times y]_{补}$。

(5) 用不恢复余数法计算$[x/y]_{原}$的商和余数。

【分析解答】

先将 x 和 y 转换为二进制数。$x=10=+01010B$,$y=-6=-00110B$。

(1) $[x]_{补}=001010B$,$[y]_{补}=111010B$,$[-y]_{补}=000110B$。$[x+y]_{补}=[x]_{补}+[y]_{补}=001010B+111010B=000100B$,因此,$x+y=4$。$[x-y]_{补}=[x]_{补}+[-y]_{补}=001010B+000110B=010000B$,因此,$x-y=+16$。

(2) $[x]_{原}=001010B$,$[y]_{原}=100110B$。将符号和数值部分分开处理。乘积的符号为$0\oplus1=1$,数值部分采用无符号数乘法算法计算 01010B×00110B 的乘积。原码一位乘法过程描述如下:初始部分积为0,在乘积寄存器前增加一个进位位。每次循环首先根据乘数寄存器中最低位决定+X 还是+0(这里 X 为被除数 001010B),然后将得到的新进位、新部分积和乘数寄存器中的部分乘数一起逻辑右移一位。共循环 5 次,最终得到一个 10 位无符号数表示的乘积 0000111100B。所以,$[x\times y]_{原}=10000111100B$,因此,$x\times y=-60$。若结果取 6 位原码,则因为高 5 位 00001 是一个非零数,所以结果溢出,即$[x\times y]_{原}\neq111100B$。验证:6 位原码的表示范围为$-31\sim+31$,显然乘积$-60$不在其范围内,结果应该溢出。(过程略)

(3) $[x]_{补}=001010B$,$[-x]_{补}=110110B$,$[y]_{补}=111010B$。采用 Booth 算法时,符号和数值部分一起参加运算,最初在乘数后面添 0,初始部分积为 0。每次循环先根据乘积寄存器中最低两位决定执行+X、-X 还是+0 操作(这里 X 为被乘数 001010B),然后将得到的新的部分积和乘数寄存器中的部分乘数一起算术右移一位。-X 采用$+[-x]_{补}$的方式进行。共循环 6 次。最终得到一个 12 位补码表示的乘积 1111 1100 0100B,所以,$[x\times y]_{补}=1111\ 1100\ 0100B$,因此,$x\times y=-60$。若结果取 6 位补码,则根据乘积低 6 位 000100 的符号位为 0,而高 6 位为 111111,不等于全 0,说明结果溢出。即$[x\times y]_{补}\neq000100B$。验证:6 位补码的表示范围为$-32\sim+31$,显然$-60$不在其范围内,结果应该溢出。(过程略)

(4) $[x]_{补}=001010B$,$[-x]_{补}=110110B$,$[y]_{补}=111010B$。采用 MBA 算法时,符号和数值部分一起参加运算,最初在乘数后面添 0,初始部分积为 0,并在部分积前加一位符号位0。每次循环先根据乘积寄存器中最低 3 位决定执行+X、+2X、-X、-2X 还是+0 操作(这里 X 为被乘数 001010B),然后将得到新的部分积和乘数寄存器中的部分乘数一起算术右移两位。-X 和-2X 分别采用$+[-x]_{补}$和$+2[-x]_{补}$的方式进行。共循环 3 次。最终得到一个 12 位补码表示的乘积 1111 1100 0100B,所以,$[x\times y]_{补}=1111\ 1100\ 0100B$,因此,$x\times y=-60$。(过程略)

(5) $[x]_{原}=001010B$,$[y]_{原}=100110B$。将符号和数值部分分开处理。商的符号为$0\oplus1=1$,数值部分采用无符号数除法算法计算 01010B 和 00110B 的商和余数。无符号数不恢复余数除法过程描述如下:初始中间余数为 0000 0001 0100B,其中,最高位为添加的符号位,用于判断余数是否大于等于 0;最后一位 0 为第一次上的商,该位商只是用于判断结果是否溢出,不包含在最终的商中。因为结果肯定不溢出,所以该位商可以直接上 0,并先做一次-Y 操作得到第一次中间余数(这里 Y 为被除数 000110B),然后进入循环。每次循环首先将中间余数和商一起左移一位,然后根据上一次上的商(或余数的符号)决定执行+Y

还是$-Y$操作，以得到新的中间余数，最后根据中间余数的符号确定上商为0还是1。$-Y$采用$+[-y]_{补}$的方式进行。整个循环内执行的要点是“正、1、减；负、0、加”。共循环5次。最终得到一个6位无符号数表示的商0 00001B和余数00100B，其中第一位商0必须去掉，添上符号位后得到最终的商的原码表示为1 00001B，余数的原码表示为0 00100B。因此，x/y的商为-1，余数为4。（过程略）

14. 假设浮点数的阶码和尾数均采用补码表示，且位数分别为5位和7位（均含2位符号位，即变形补码）。若有两个数$x=2^7\times15/16$，$y=2^5\times3/8$，要求用浮点加法计算$x+y$的最终结果。

【分析解答】

先将两个数的尾数部分变成分母为32的形式，即$x=2^7\times30/32$，$y=2^5\times12/32$，得到x和y的浮点数表示为x：00.11110B，00111B、y：00.01100B，00101B；然后进行浮点数加/减运算。y对阶后为00.00011B，00111B，因此尾数相加结果为00.11110B＋00.00011B＝01.00001B。该尾数形式需要右规，即尾数右移一位，若采用“0舍1入”舍入规则，右规后尾数为00.10001B，阶码加1后为01000B，因此，右规后结果为00.10001B，01000B。最后对该结果进行溢出判断，显然，阶码的两个符号位不同，故结果溢出。

15. 假设有两个实数x和y，$x=-68$，$y=-8.25$，它们被定义为float型变量，x和y分别存放在寄存器A和B中。另外，还有两个寄存器C和D。A、B、C、D都是32位寄存器。请回答下列问题（要求最终用十六进制表示二进制序列）。

(1) 寄存器A和B中的内容分别是什么？

(2) 若x和y相加后的结果存放在寄存器C中，则寄存器C中的内容是什么？

(3) 若x和y相减后的结果存放在寄存器D中，则寄存器D中的内容是什么？

【分析解答】

通常float型数据用IEEE 754单精度浮点数格式表示。

(1) $x=-68=-1000100B=-1.0001B\times2^6$，因此，符号位为1，阶码$Ex$为1000 0101B，尾数小数部分为000 1000 0000 0000 0000 0000B，浮点数表示形式为1 1000 0101 000 1000 0000 0000 0000 0000B，十六进制形式为C288 0000H。$y=-8.25=-1000.01B=-1.00001B\times2^3$，因此，符号位为1，阶码$Ey$为1000 0010B，尾数小数部分为000 0100 0000 0000 0000 0000B，浮点数表示形式为1 1000 0010 000 0100 0000 0000 0000 0000B，十六进制形式为C104 0000H。因此，寄存器A和B中的内容分别是C288 0000H、C104 0000H。

(2) 两个浮点数相加的步骤如下。

① 对阶。Ex=1000 0101B，Ey=1000 0010B，$[\Delta E]_{补}=Ex+[-Ey]_{补}$=1000 0101B＋0111 1110B＝0000 0011B，$\Delta E=+3$，所以对y进行对阶。对阶后，$y=-0.00100001B\times2^6$。即y的浮点数表示为1 1000 0101 (0) 001 0000 1000 0000 0000 0000B。

② 尾数相加。x的尾数为$-1.$ 000 1000 0000 0000 0000 0000B，y的尾数为$-0.$ 001 0000 1000 0000 0000 0000B。原码加法运算规则为“同号求和，异号求差”。因两数符号相同，故做加法，结果为$-1.$001 1000 1000 0000 0000 0000B。因此，$x+y$的结果为$-1.$001 1000 1B$\times2^6$，符号位为1，尾数小数部分为001 1000 1000 0000 0000 0000B，阶码为127＋6＝1000 0101B，浮点数表示为1 1000 0101 001 1000 1000 0000 0000 0000B，转换为十

六进制形式为 C298 8000H。因此，寄存器 C 中的内容是 C298 8000H。

（3）两个浮点数相减的步骤同加法，对阶的结果也一样，只是尾数相减。原码减法运算规则为“同号求差，异号求和”。因两数符号相同，故做减法。数值部分由被减数加上减数的补码（各位取反，末尾加 1）得到，即：

$$
\begin{array}{r}
1.\ 000\ 1000\ 0000\ 0000\ 0000\ 0000 \\
+\quad 1.\ 110\ 1111\ 1000\ 0000\ 0000\ 0000 \\
\hline
1\ 0.\ 111\ 0111\ 1000\ 0000\ 0000\ 0000
\end{array}
$$

最高数值位产生进位，表明所得数值位正确，且结果的符号取被减数的符号，即结果为负数。因此，x 减 y 的结果为 $-0.11101111B\times 2^6=-1.1101111B\times 2^5$。即符号位为 1，尾数小数部分为 110 1111 0000 0000 0000 0000B，阶码为 127＋5＝1000 0100B，浮点数表示为 1 1000 0100 110 1111 0000 0000 0000 0000B，转换为十六进制形式为 C26F 0000H。因此，寄存器 D 中的内容是 C26F 0000H。

16. 在 IEEE 754 浮点数运算中，当结果的尾数出现什么形式时需要进行左规？什么形式时需要进行右规？如何进行左规？如何进行右规？

【分析解答】

（1）对于结果为 $\pm 1x.xx\cdots x$ 的情况，需要进行右规。即尾数 M_b 右移一位，阶码 E_b 加 1。右规操作可以表示为 $M_b\leftarrow M_b\times 2^{-1}, E_b\leftarrow E_b+1$。右规时注意以下两点：(a)尾数右移时，最高位“1”被移到小数点前一位作为隐藏位，最后一位移出时，要考虑舍入；(b)阶码加 1 时，直接在末位加 1。

（2）对于结果为 $\pm 0.00\cdots 01x\cdots x$ 的情况，需要进行左规。即尾数 M_b 逐次左移，阶码 E_b 逐次减 1，直到将第一位“1”移到小数点左边。假定 k 为结果中“±”和左边第一个 1 之间连续 0 的个数，则左规操作可以表示为 $M_b\leftarrow M_b\times 2^k, E_b\leftarrow E_b-k$。左规时注意以下两点：(a)尾数左移时数值部分最左 k 个 0 被移出，因此，相对来说，小数点右移了 k 位。因为进行尾数相加时，默认小数点位置在第一个数值位（即隐藏位）之后，所以小数点右移 k 位后被移到了第一位 1 后面，这个 1 就是隐藏位。(b)执行 $E_b\leftarrow E_b-k$ 时，每次都在末位减 1，一共减 k 次。

17. 在 IEEE 754 浮点数运算中，如何判断浮点运算的结果是否溢出？

【分析解答】

因为尾数溢出时，可通过右规操作进行纠正。所以，浮点运算结果是否溢出，并不以尾数溢出来判断，而主要看阶码是否溢出。在进行规格化、尾数舍入和浮点数的乘/除运算过程中，都需要对阶码进行加、减运算，因此在这些操作过程中，可能会发生阶码上溢或阶码下溢。阶码上溢时，说明结果的数值太大，无法表示，是真正的溢出；阶码下溢时，说明结果数值太小，可以将结果近似为 0。

18. 假设浮点数格式为：阶码是 4 位移码，偏置常数为 8，尾数是 6 位补码（采用双符号位），用浮点运算规则分别计算在不采用任何附加位和采用 2 位附加位（保护位、舍入位）的情况下以下各式的值（假定对阶和右规时采用就近舍入到偶数方式）。

（1）$(15/16)\times 2^7+(2/16)\times 2^5$　　（2）$(15/16)\times 2^7-(2/16)\times 2^5$

（3）$(15/16)\times 2^5+(2/16)\times 2^7$　　（4）$(15/16)\times 2^5-(2/16)\times 2^7$

【分析解答】

将上述各式中的数据用相应的变量 A、B、C、D 代替。

$A=(15/16)\times2^7=0.1111B\times2^7$，$[A]_{浮}=00.1111B,1111B$。

$B=(2/16)\times2^5=0.0010B\times2^5=0.1000B\times2^3$，$[B]_{浮}=00.1000B,1011B$。

$C=(15/16)\times2^5=0.1111B\times2^5$，$[C]_{浮}=00.1111B,1101B$。

$D=(2/16)\times2^7=0.0010B\times2^7=0.1000B\times2^5$，$[D]_{浮}=00.1000B,1101B$。

不采用任何附加位时的计算结果如下。

(1) 计算 $A+B$：$[\Delta E]_{补}=[E_A]_{移}+[-[E_B]_{移}]_{补}=1111B+0101B=0100B(\bmod 2^4)$，因此 $\Delta E=4$，故需对 B 进行对阶，因为不采用任何附加位，所以，B 的尾数右移 4 位后直接舍去 1000，又因为“1000”是中间值，因此尾数取偶数 00.0000B，故对阶后结果为 $[B]_{浮}=00.0000B,1111B$。由于 B 的尾数为 0，因此，$[A+B]_{浮}=[A]_{浮}=00.1111B,1111B$。故 $A+B=A=(15/16)\times2^7$。

(2) 计算 $A-B$：对阶结果与(1)相同，故$[A-B]_{浮}=[A]_{浮}=00.1111B,1111B$。故 $A-B=A=(15/16)\times2^7$。

(3) 计算 $C+D$：$[\Delta E]_{补}=[E_C]_{移}+[-[E_D]_{移}]_{补}=1101B+0011B=0000B(\bmod 2^4)$，因此 $\Delta E=0$，故无须对阶。尾数直接加：$[M_C]_{补}+[M_D]_{补}=00.1111B+00.1000B=01.0111B$，因为补码的两个符号位不同，所以尾数溢出，需要右规。右规时需对尾数进行舍入，阶码加 1。舍入最后一位的“1”是中间值，因此尾数取偶数 00.1100B，阶码 1101B 加 1 后为 1110B，所以，$[C+D]_{浮}=00.1100B,1110B$。故 $C+D=(12/16)\times2^6$。

(4) 计算 $C-D$：对阶结果与(3)相同。尾数直接减：$[M_C]_{补}+[-M_D]_{补}=00.1111B+11.1000B=00.0111B$。显然，尾数需左规。左规时，尾数左移一位，阶码减 1。因此，最终尾数为 00.1110B，阶码 1101B 减 1 后为 1100B。因此，$[C-D]_{浮}=00.1110B,1100B$，故 $C-D=(14/16)\times2^4$。

采用两位附加位时的计算结果如下。

(1) 计算 $A+B$：$[\Delta E]_{补}=[E_A]_{移}+[-[E_B]_{移}]_{补}=1111B+0101B=0100B(\bmod 2^4)$，因此 $\Delta E=4$，故需对 B 进行对阶，对阶后结果为$[B]_{浮}=00.0000\ 10B,1111B$。尾数相加结果为$[M_A]_{补}+[M_B]_{补}=00.1111\ 00B+00.0000\ 10B=00.1111\ 10B$，因此，$[A+B]_{浮}$为 00.1111 10B,1111B。最后对尾数附加位 10 进行舍入，因为舍入的是中间值，所以尾数结果强迫为偶数，即尾数末位加 1，得尾数为 01.0000B，因此，尾数需右规为 00.1000B，同时，阶码 1111B 加 1，产生阶码上溢，因而导致结果溢出。因此，$A+B$ 的结果溢出。

(2) 计算 $A-B$：对阶结果与(1)相同。尾数相减结果为$[M_A]_{补}+[-M_B]_{补}=00.1111\ 00B+11.1111\ 10B=00.1110\ 10B$，因此，$[A-B]_{浮}=00.1110\ 10B,1111B$。最后对尾数附加位 10 进行舍入，因为舍入的是中间值，所以尾数结果强迫为偶数，得尾数为 00.1110B，因此，$[A-B]_{浮}=00.1110B,1111B$。故 $A-B=(14/16)\times2^7$。

(3) 计算 $C+D$：$[\Delta E]_{补}=[E_C]_{移}+[-[E_D]_{移}]_{补}=1101B+0011B=0000B(\bmod 2^4)$，因此 $\Delta E=0$，故无须对阶。尾数直接加：$[M_C]_{补}+[M_D]_{补}=00.1111\ 00B+00.1000\ 00B=01.0111\ 00B$，因为补码的两个符号位不同，所以尾数溢出，需要右规。右规时需对尾数进行舍入，阶码加 1。舍入的“100”是中间值，因此尾数取偶数 00.1100B，阶码 1101B 加 1 后为 1110B，所以，$[C+D]_{浮}=00.1100B,1110B$。故 $C+D=(12/16)\times2^6$。

(4) 计算 $C-D$：对阶结果与(3)相同。尾数直接减：$[M_C]_{补}+[-M_D]_{补}=00.1111\ 00B+11.1000\ 00B=00.0111\ 00B$。显然，尾数需左规。左规时，尾数左移一位，阶码减 1。因此，

最终尾数为00.1110B，阶码1101B减1后为1100B。因此，$[C-D]_{浮}$=00.1110B，1100B，故$C-D=(14/16)\times 2^4$。

19. 采用IEEE 754单精度浮点数格式计算下列表达式的值。

(1) 0.75+(−65.25)　　(2) 0.75−(−65.25)

【分析解答】

x=0.75=0.11B=1.1B×2^{-1}，y=−65.25=−1000001.01B=−1.00000101B×2^6。用IEEE 754单精度格式表示为$[x]_{浮}$=0 01111110 10…0B，$[y]_{浮}$=1 10000101 000001010…0B。即x的阶码E_x=01111110B，x的尾数M_x=0(1).10…0B，y的阶码E_y=10000101B，y的尾数M_y=1(1).000001010…0B。尾数M_x和M_y的小数点前面有两位，第一位为数符，第二位加了括号，是隐藏位"1"。以下是机器中浮点数加/减运算过程(假定保留2位附加位：保护位和舍入位)。

(1) 0.75+(−65.25)

① 对阶：$[\Delta E]_{补}=E_x+[-E_y]_{补}$=0111 1110B+0111 1011B=1111 1001B(mod 2^8)，$\Delta E=-7$，故需对x进行对阶，结果为$E_x=E_y$=10000101B，M_x=00.000000110…0 **00**B，也即将x的尾数M_x右移7位，符号不变，数值高位补0，隐藏位右移到小数点后面，最后移出的2位保留作为附加位(粗体字部分)。

② 尾数相加：$M_b=M_x+M_y$=00.000000110…0 **00**B+11.000001010 …0 **00**B。根据原码加/减法运算规则，得M_b=11.000000100…0 **00**B。上式尾数中最左边第一位是符号位，其余都是数值部分，尾数最后两位是附加位。

③ 规格化：根据所得尾数的形式，数值部分最高位为1，所以不需要进行规格化。

④ 舍入：将结果的尾数M_b中最后两位附加位舍入，从本例来看，不管采用什么舍入法，结果都一样，都是把最后两个0去掉，得M_b=11.000000100…0B。

⑤ 溢出判断：在上述阶码计算和调整过程中，没有发生"阶码上溢"和"阶码下溢"的问题。

最终结果为E_b=10000101B，M_b=1(1).00000010…0B，即−1.0000001B×2^6=−64.5。

(2) 0.75−(−65.25)

① 对阶：同上述(1)中对阶过程一样。

② 尾数相减：$M_b=M_x-M_y$=00.000000110…0 **00**b−11.000001010…0 **00**B。根据原码加/减法运算规则，得M_b=01.00001000…0 **00**B。

③ 规格化：根据所得尾数的形式，数值部分最高位为1，不需要进行规格化。

④ 舍入：把结果的尾数M_b中最后两位附加位舍入掉，得M_b=01.00001000…0B。

⑤ 溢出判断：在上述阶码计算和调整过程中，没有发生"阶码上溢"和"阶码下溢"的问题。

最后结果为E_b=10000101，M_b=0(1).00001000…0B，即+1.00001B×2^6=+66。

20. 假定十进制数用8421 NBCD码表示，采用十进制加法运算计算下列表达式的值，并讨论在十进制BCD码加法运算中如何判断溢出。

(1) 234+567　　(2) 548+729

【分析解答】

(1) 计算234+567时，两个加数分别是0010 0011 0100B和0101 0110 0111B，送到12

位无符号加法器中得到和为 0111 1001 1011B，其中，最低 4 位需进行“+6”校正，得到结果为 0111 1010 0001B，此时，中间 4 位又需要“+6”校正，得最终结果为 1000 0000 0001B，转换为十进制数为 801。

(2) 计算 548+729 时，两个加数分别为 0101 0100 1000B 和 0111 0010 1001B，送到 12 位无符号加法器中得到和为 1100 0111 0001B，最高 4 位和最低 4 位需“+6”校正，得到低 12 位结果为 0010 0111 0111B，并产生进位 1，因而结果溢出。但如果采用 4 位 BCD 码表示（即在 16 位加法器中运算），则结果不会溢出，此时得到最终的和为 0001 0010 0111 0111B，即十进制数 1277。

21. 假定十进制数用 8421 NBCD 码表示，十进制运算 673－356 可以采用 673 加上－356 的模 10 补码实现。画出实现上述操作的 3 位十进制数的 BCD 码减法运算线路。

【分析解答】

计算 673－356 时，先通过将 356“各位取反、末位加 1”得到－356 的模 10 补码表示，为 0110 0100 0100B，然后与被加数 673 相加，因此，无符号加法器的两个加数输入端分别为 0110 0111 0011B 和 0110 0100 0100B。这样加法器的输出为 1100 1011 0111B，其中高 4 位和中间 4 位需“+6”校正，得到低 12 位结果为 0011 0001 0111B，同时产生进位 1。在 BCD 码减法运算中，若最高位有进位，则说明结果为正；若最高位没有进位，则说明结果为负，需将数值部分“各位取反，末尾加 1”，以得到最终结果。因此，本题结果为十进制数 317。

BCD 码减法运算电路通常在 BCD 码加法器基础上实现，可在第二个加数输入端加一个十进制数字“求补”电路，并在输出端处增加一个对进位进行判断和对结果“求补”的电路。实现 3 位 BCD 码减法运算的电路如图 3.1(a)所示。图中 3 位被减数为 $x_2x_1x_0$，3 位减数为 $y_2y_1y_0$，结果的符号为 s，$s=0$ 表示结果为正数，$s=1$ 表示结果为负数，结果的数值部分为 $z_2z_1z_0$。“3 位 BCD 码求补电路”如图 3.1(b)所示，通过“各位取反、末位加 1”得到补码。“一位 BCD 码取反电路”如图 3.1(c)所示，通过“加 6 取反”得到一位 BCD 码的反码。

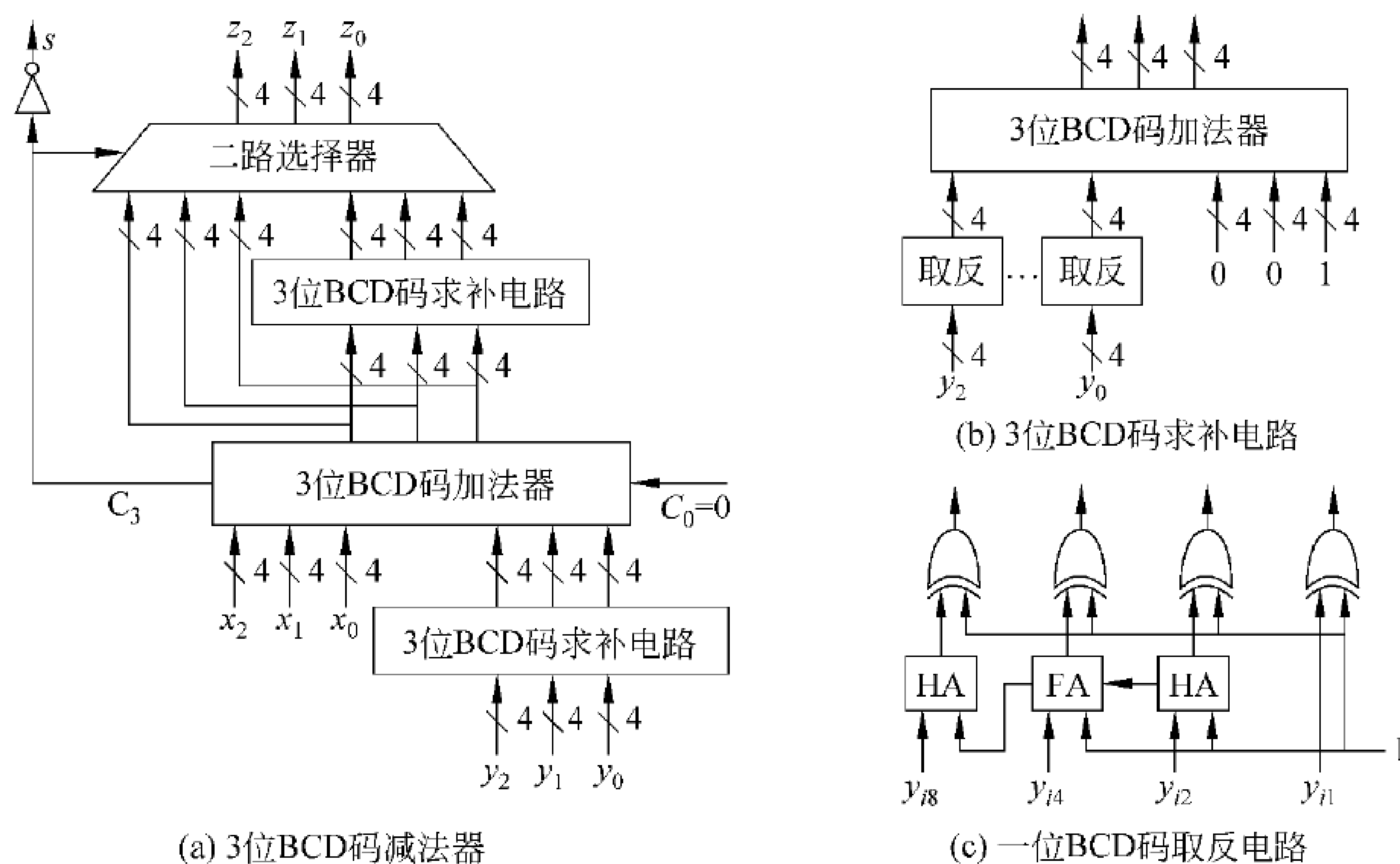


图 3.1　3 位 BCD 码减法运算线路

第 4 章 存储器分层体系结构

4.1 教学目标和内容安排

主要教学目标：

使学生掌握构成存储器分层体系结构的几类存储器的工作原理和组织形式。要求学生深刻理解程序访问局部性的意义，学会利用时间局部性和空间局部性编写高效的程序；了解指令执行过程中访问指令和访问数据的整个过程，以及存储访问过程中硬件和软件的分工和联系，并深刻理解提高各种访问命中率的意义；了解虚拟存储管理的必要性和实现思路，为学习操作系统中的存储管理等内容打下坚实基础。

基本学习要求：

(1) 了解存储器的各种分类方式。

(2) 了解如何构建存储器的层次化结构。

(3) 深刻理解程序访问的局部化特性。

(4) 熟悉主存储器的基本逻辑结构。

(5) 了解 SRAM 和 DRAM 芯片的内部结构。

(6) 了解半导体随机存取存储器的组织方式。

(7) 了解各种只读存储器的特点。

(8) 了解存储器芯片扩展技术及其与 CPU 的连接方式。

(9) 了解加快存储访问速度的几种措施。

(10) 了解多体交叉编址存储器的基本原理。

(11) 掌握 cache 的基本原理与实现方式，包括三种映射方式、替换算法、写策略等。

(12) 理解为何采用虚拟存储管理方式。

(13) 理解什么是虚拟地址和虚拟地址空间。

(14) 掌握虚拟地址向物理地址转换的基本原理与实现技术。

(15) 了解页表的功能和页表项的内容。

(16) 了解“缺页”异常的发现和处理过程。

(17) 掌握 TLB(快表)的结构和实现技术。

(18) 掌握一次存储访问的全过程，并深刻理解在此过程中硬件与软件之间的分工协作方式。

本章主要包含3个核心内容：半导体随机访问存储器、cache和虚拟存储器，并阐述了如何以这3个核心内容为基础，构建存储器的层次化体系结构框架。

对于半导体随机访问存储器，可以按照“记忆单元→存储阵列→存储器芯片→存储模块(内存条)→存储器”的顺序，采用由点到面的组织方式，着重讲清楚SRAM和DRAM两类存储器的结构、特点和用途，以及存储芯片的扩展和连接技术。有关存储芯片的扩展和连接技术方面的知识，有助于对总线、数据的存放顺序和对齐方式等许多概念的理解。此外，同步存储器芯片(如SDRAM芯片)的概念与后面CPU设计和总线设计的内容相关。在课时受限的情况下，对于记忆单元的存储和读写原理、存储芯片的读写周期、DRAM的刷新、只读存储器等内容只要概要说明即可，因为这些不属于主干内容，也比较独立，对其他内容的学习影响不大。

对于cache，首先，应着重讲清楚程序访问的局部性，因为程序的时间局部性和空间局部性是提出并实现cache的基础，对这些内容的深刻理解，也有助于编写高效的程序。通过具体程序示例，可以将程序访问的局部性特点讲深讲透。cache和主存之间的映射关系可能是难点部分，主教材①中例4.4、例4.7和例4.8是针对相同主存大小和相同cache行数的3种不同映射关系的例子，课堂教学中，可以跳过主教材中对映射关系的形式化描述，而直接通过这3个例子来说明不同的映射关系。这样，使学生能够较快地掌握不同映射关系的不同实现方式和访存过程。对于主教材中的例4.10，可通过详细讲解几个关键单元(如第0、1、63、64等单元)的访问，使学生能够深入了解CPU的访存过程和替换算法。如果课时受限，cache性能评估、cache结构举例，以及影响cache性能的因素等内容可以跳过或仅作简单讲解。

对于虚拟存储器，着重讲清楚请求分页的思想、虚拟地址空间的概念、页表的结构、地址转换过程和快表的概念。主教材中图4.43和图4.44反映了CPU进行一次存储访问的过程，结合对这两张图的讲解和对表4.1的解读，可以加深学生对CPU访存过程的理解，特别是加深对软件和硬件分工协作过程的了解，从而加深对完整的存储器层次化结构体系的理解。如果课时受限，有关进程与进程的上下文、存储器管理概述、存储保护等可跳过或仅作简单讲解。

4.2 主要内容提要

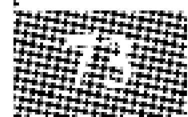
1. 存储器的分类

存储器按存取方式分为随机存取存储器、顺序存取存储器、直接存取存储器和相联存取存储器；按存储介质分为半导体存储器、磁表面存储器和光盘存储器；按信息的可更改性分为可读可写和只读存储器；按断电后可否保存来分，分为易失性和非易失性存储器；按功能、容量、速度3个方面来分，分成寄存器、高速缓存(cache)、主存储器(内存)、辅助存储器(外存)和海量后备存储器。

2. 存储器的分层结构

因为每一种单独的存储器都不可能又快，又大，又便宜，为了构建这种理想的存储器系

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)



统,计算机中采用了一种层次化的存储器体系结构。按照离 CPU 由近到远、速度从快到慢、容量从小到大、价格从贵到便宜的顺序,将不同的存储器设置在计算机中,这样的顺序是寄存器→cache→主存→磁盘→光盘和磁带。

3. 半导体随机存取存储器的组织

主存空间的 RAM 区由若干内存条组成,每个内存条上有若干存储器芯片,存储器芯片由用于存储信息的存储阵列外加地址缓存器、地址译码器、读写控制电路等组成,每个存储阵列由若干行和若干列构成,每个行、列交叉处是一个记忆单元(存储元),每个记忆单元用来存储一位二进位 0 或 1。根据记忆单元结构的不同分为 SRAM 芯片和 DRAM 芯片两种,SRAM 芯片的记忆单元采用 6 管静态 MOS 管存储电路,其功耗大、集成度低,但速度快,无须再生和刷新,适合构作高速小容量的存储器,如 cache;DRAM 芯片的记忆单元采用单管动态 MOS 管存储电路,因为只用一个 MOS 管,所以功耗小、集成度高,但由于靠电容储存电荷和充放电来存储和读写信息,所以速度慢,并需定时刷新,适合构作慢速大容量的存储器,如主存。

4. 只读存储器

只读存储器中的信息用特殊方式写入,一经写入,就可长久保存,是非易失性存储器。其存取方式也为随机存取方式。主要用于存放固定信息,如微程序、BIOS、引导程序或嵌入式系统中固化的程序和数据等。有 MROM、PROM、EPROM、EEPROM、Flash ROM 等类型。

5. 存储器芯片及其与 CPU 的连接

RAM 芯片分为字片式和位片式两种。通常 DRAM 芯片都是位片式,多位 DRAM 芯片采用多个位平面构成,每个位平面是一个二维的存储阵列。行地址和列地址共用同一组地址引脚,称为行列地址线复用。为提高存储器芯片的读写速度,DRAM 芯片内通常会有一个用 SRAM 实现的行缓存。

存储器芯片和 CPU 之间通过总线相连,总线中包括地址线、数据线和控制线。地址线的连接需要考虑芯片在字方向上的扩展,采用芯片内连续编址方式时,低位用于芯片内地址,高位用于片选逻辑,片选信号译码器的输出连到芯片的片选信号引脚上;数据线的连接需要考虑芯片在位方向上的扩展,分别连到位扩展的芯片上。

6. 主存的主要技术指标

包括存储容量、存取时间、存储周期和存储器带宽。存储容量是指某计算机实际配置的容量,通常,它小于最大可配置容量(主存地址空间大小);存取时间指执行一次读操作或写操作的时间,分读出时间和写入时间两种;存储周期指存储器进行连续两次独立的读或写操作所需要的最小时间间隔,它通常大于存取时间;存储器带宽指单位时间内从存储器读出或写入存储器的最大信息量。

7. 多模块存储器

采用多模块存储器的目的是为了提高访存速度,通过多个存储模块并行工作可以达到目的。为使多个模块并行工作,每个模块除了有各自独立的存储阵列以外,还必须有独立的地址缓存器、数据缓存器、地址译码器和读写控制电路等。

有连续编址和交叉编址两种组织方式。连续编址方式下,按高位地址划分模块,地址在一个存储模块内连续编号,因此同一个访存请求内的信息在同一个模块中,无法并行。只有

同时有多个访存请求时才能并行；交叉编址方式下，按低位地址划分模块，地址在所有存储模块之间交叉编号，因此同一个访存请求内的信息分布在不同模块中，可并行访问所有模块，因而可加快访问速度。

8. 高速缓存(cache)

cache是在CPU和主存之间设置的高速小容量的存储器。引入cache的目的是为了提高访存速度。与多模块存储器通过并行来提高速度不同，cache之所以能提高速度，是因为程序执行时代码和数据的存储访问具有局部性特点。程序访问的局部性特点体现在两个方面：时间局部性和空间局部性。时间局部性指某个单元在一个很短的时间段内被重复访问的可能性很大；空间局部性指某个单元被访问后其周围单元不久也将被访问的可能性很大。这样，只要将刚被访问的单元及其邻近单元一起复制到cache中，那么，在最近一段时间内CPU访问的信息都可以在cache中访问到，而不需要访问慢速的主存。

实现cache时需要解决一系列问题，例如，将主存中的一个局部信息块装入cache时，信息块大小多大？装入到cache的何处？CPU如何根据主存地址找到cache中相应的信息？cache装满的情况下又要复制新的主存块到cache时，原来在cache中的哪些主存块应被替换出来？写信息时如何保证主存中和cache中的同一个信息块完全一致？对于上述问题的简要回答如下。

(1) cache和主存间的映射关系

将主存地址空间划分成大小相等的信息块，从0开始给每个块编号。cache由若干行组成，每一行中有一个用于存放主存块的槽，其大小与主存块大小一样，cache行也从0开始编号。在将主存块复制到cache行时，主存块号和cache行号之间可采用直接映射、全相联映射和组相联映射3种映射关系。

直接映射时，每个主存块对应一个固定的cache行，其映射关系为：

cache行号＝主存块号 mod cache行数

此时，主存地址划分为标记、cache行号(行索引)和块内地址3个字段。

全相联映射时，每个主存块可复制到任何一个cache行中，主存地址划分为标记和块内地址两个字段。

组相联映射时，cache分若干组，每组有多行，各主存块存放到固定组的任意行中，其映射关系为：

cache组号＝主存块号 mod cache组数

此时主存地址划分为标记、cache组号(组索引)和块内地址3个字段。

(2) CPU访存过程

CPU给出主存地址后，首先根据映射方式对主存地址进行划分，根据中间的行索引或组索引的值，确定将主存地址高位部分的标记字段与哪些cache行中的标记进行比较。显然，对于直接映射，只需比较一个cache行；对于全相联映射，则需与所有行进行比较；对于组相联映射，则与组内所有行比较。若存在某个cache行中的标记与主存地址中的标记字段相等，并且该行中的有效位为1，则访问命中，此时，根据主存地址中低位部分的块内地址访问该行中相应的信息；若所有行中的标记都不等于主存地址中的标记字段，或有相等的行但对应的有效位为0，则访问不命中(缺失)，此时，需要将该主存地址所在的块从主存取到cache，并根据主存块的位置在cache行中置标记，且置有效位为1。

(3) 替换算法

当需要调入一个新的主存块而对应的cache行全满时，需要将这些cache行中某个主存块替换出来。常用的替换算法有先进先出(FIFO)、最近最少用(LRU)等。FIFO算法的基本思想是，总是把最先调到cache的那个主存块淘汰掉；LRU算法的基本思想是，总是把最近最少用到的那个主存块淘汰掉。

(4) 写策略(一致性问题)

CPU执行写操作时，为了保证主存和cache中同一个主存块的一致性，可采用回写法(Write Back)和全写法(Write Through)两种写策略。回写法的基本思想是，暂时只写cache，替换时一次性将cache中的主存块写回主存；全写法的基本思想是，每次写cache的同时也写主存，为了加快写的过程，可在cache和主存间加一个写缓存(Write Buffer)。

当写不命中时，有写分配法(Write Allocate)和非写分配法(Not Write Allocate)两种方式。采用写分配法时，需要分配一个cache空行，以将主存块复制到cache；采用非写分配法时，不将主存块复制到cache。因此，回写策略下，一定采用写分配法，而全写策略下，两种分配方式都可以采用。

(5) 主存块大小的选择

主存块大小是主存和cache之间进行信息交换的基本单位，主存块大小与命中率和缺失损失关系极大，因而块大小的选择非常重要。主存块太小，则不能很好地利用空间局部性，进而影响命中率；主存块太大，则增加主存块的读取时间，即缺失损失变大，而且，由于块变大，使得cache行数减少，映射到同一个cache行的主存块数增加，进而会使缺失率上升。

9. 虚拟存储器

虚拟存储管理是现代计算机系统中普遍采用的存储管理方式。在采用虚拟存储管理的计算机系统中，每个进程具有一个一致的、极大的、私有的虚拟地址空间，虚拟地址空间按等长的页来划分，主存也按等长的页框划分。进程执行时将当前用到的页面装入主存，其他暂时不用的部分放在磁盘上，通过页表建立虚拟页和主存页框之间的对应关系。对于不在主存的页面，在页表中记录其在磁盘上的地址。在指令执行过程中，由特殊的硬件MMU和操作系统一起实现存储访问。

虚拟存储器的实现方式有分页式、分段式和段页式3种。CPU执行指令时，通过指令寻址方式计算得到的有效地址通常是一个虚拟地址(即逻辑地址)。CPU中的地址转换部件根据虚拟地址中的虚页号，找到对应的页表项，然后通过页表项得到该虚页号对应的页框号(即物理页号、实页号)，最后将它和页内地址拼接得到物理地址(即主存地址、实地址)。

每个进程有一个页表，每个页表项由有效(装入)位、使用位、修改位、存取权限位、主存页框号或磁盘地址等组成。在地址转换过程中，若对应页表项中的有效位为0，则说明该页面不在主存中，即"缺页"，此时，CPU调出操作系统的缺页处理程序执行，该程序从磁盘读入所需页面到主存，并修改页表。缺页处理后，必须回到原来发生缺页的指令重新执行。

为了减少从主存访问页表的次数，通常将常用页表项放在CPU的一个高速缓存中，这个高速缓存被称为TLB(快表)。

可以利用虚拟存储管理机制进行存储保护，主要有地址越界和访问越权两种内存保护错误，通常称它们为访问违例或存储器访问异常。

图4.1给出了CPU通过TLB和页表进行地址转换，并根据转换得到的主存地址进行

cache 访问的全过程。图中 TLB 和 cache 都采用组相联映射方式。

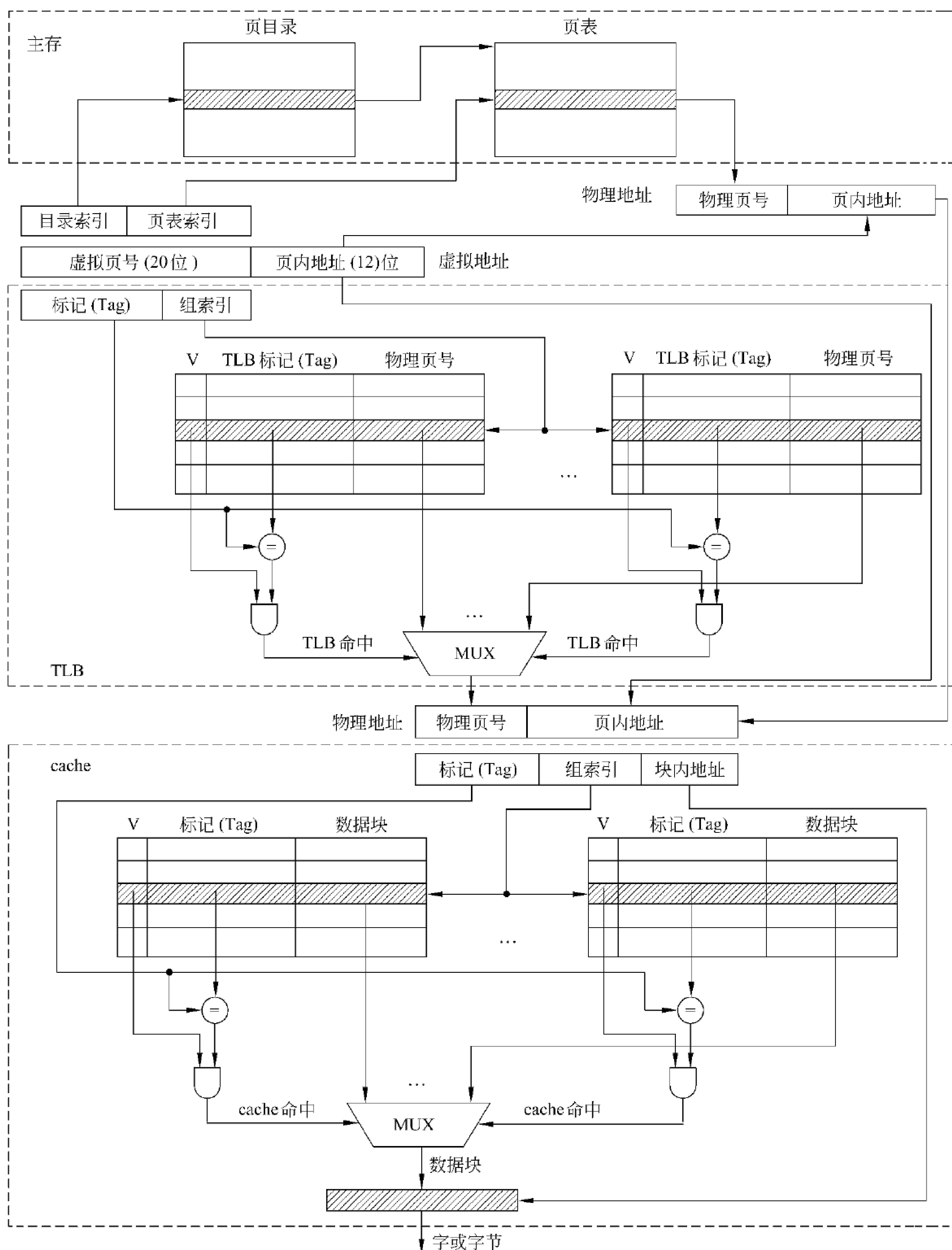


图 4.1 地址转换和 cache 访问过程

4.3 基本术语解释

随机访问存储器(Random Access Memory,RAM)

其特点是按地址访问信息。对于一个RAM芯片来说,所有单元的地址位数一样,所以每个单元的地址译码所用时间一样。从这个角度来说,这种存储器中每个单元的存取时间与存储单元的物理位置无关。

相联存储器(Content Addressed Memory,Associate Memory,CAM)

其特点是按内容访问信息。即已知要访问信息的部分内容,通过与每个存储单元的相应内容进行比较找到需访问信息的位置,然后读写信息。

静态随机访问存储器(Static RAM,SRAM)

靠触发器的双稳态正负反馈电路存储信息,因而速度快,是非破坏性读出,但电路中元器件多,因而集成度小,适合做高速小容量的高速缓冲存储器(cache)。

动态随机访问存储器(Dynamic RAM,DRAM)

靠电容存储电荷来保存信息。若电容上存有足够多的电荷表示存“1”,电容上无电荷表示存“0”。是破坏性读出,读后需要再生,而且需要定时刷新。

刷新(Refresh)

DRAM芯片中,MOS管栅极电容上的电荷会被逐渐放电,只能维持几到几十毫秒,因此,即使电源不掉电,也会自动消失,需要定时对所有存储单元进行充/放电,以恢复原来的电荷。这个过程称为刷新。

易失性存储器(Volatile Memory)

电源掉电后,存储器中的信息全部消失,如高速缓存(cache)、主存中的RAM等都属于易失性存储器。

非易失性存储器(Nonvolatile Memory)

存储器中的信息不会因为电源掉电而消失,如只读存储器(ROM)、磁盘、光盘、闪存(Flash存储器)等都属于非易失性存储器。

记忆单元(存储基元、存储元)(Memory Cell)

具有两种稳态的能够表示二进制数码0和1的物理器件,一个记忆单元表示1位信息。

存储单元(Memory Unit)

主存中具有相同地址的那些位构成一个存储单元。因此,存储单元的宽度等于一个编址单位的长度,可以是8位、16位、32位等。现在,大多数计算机是按字节编址的,即每一个字节(8位)有一个地址,编址单位就是一个字节,所以一个存储单元的宽度(位数)是8位。

存储器地址寄存器(Memory Address Register,MAR)

CPU中用来存放存储器地址的寄存器,地址在送到总线的地址线之前,先寄存在MAR中。所以,它的宽度应该等于地址线的宽度,也等于主存地址位数,其值决定了主存最大的寻址空间。

存储器数据寄存器(Memory Data Register,MDR)

有时也称为存储器缓冲寄存器(MBR)。CPU中用来存放写入主存或从主存读出的数据的寄存器,数据在送到总线的数据线之前,或从主存读到CPU时,都先寄存在MDR中。

所以，它的宽度应该等于总线数据线的宽度。

存取时间(Access Time)

执行一次读操作或写操作的时间。分读出时间和写入时间。读出时间为从主存接收有效地址开始到数据取出有效为止的时间；写入时间是从主存接收有效地址开始到数据写入被写单元为止的时间。

存储周期(Memory Cycle Time)

存储周期 Tm 是指存储器进行连续两次独立的读或写操作所需要的最小时间间隔。

存储器带宽(Bandwidth)

每秒钟从存储器进/出信息的最大数量。假设存储周期为 50ns，每个存储周期最多可存取 64 位数据，则带宽为 1.28Gb/s。

片选信号(Chip Select Signal)

一个存储芯片的容量往往满足不了计算机对存储容量的要求，所以需要将一定数量的芯片按一定方式连接成一个完整的存储器。在访问某个字时，必须"选中"该字所在的芯片，而其他芯片不被"选中"，控制芯片是否被选中的信号即片选信号 CS。

地址引脚复用(Address Pin Multiplexing)

DRAM 芯片采用二维译码方式，为了减少引脚个数，把行地址和列地址用同一组地址引脚线分时进行传送。靠行地址选通信号和列地址选通信号来区分在地址引脚线上传送的是行地址还是列地址。

行地址选通信号(Row Address Strobe,RAS)

DRAM 芯片中，行地址选通信号有效时，说明在地址引脚线上传输的是行地址信号。

列地址选通信号(Column Address Strobe,CAS)

DRAM 芯片中，列地址选通信号有效时，说明在地址引脚线上传输的是列地址信号。

只读存储器(Read Only Memory,ROM)

这种存储器的原始信息一旦被写入，在程序执行过程中，只能对其内容进行读出，而不能写入。只读存储器通常用来存放固定不变的信息。

掩膜 ROM(Mask ROM)

由厂家在生产过程中一次形成的。即信息已经完全固化在芯片中，无法修改。结构类似于字片式 RAM，没有写入机构。这类 ROM 适合大批量生产。

PROM(Programmable ROM)

可编程只读存储器，在使用时由使用者专门一次写入，以后再也不能改变。

EPROM(Erasable PROM)

可擦除可编程只读存储器，可以用特殊的装置反复擦除和重写。一般将芯片放在紫外线下照射 15～20 分钟，信息全部擦除。

EEPROM(Electrically Erasable PROM)

电可擦除可编程只读存储器，使用电可擦除技术(加高电压擦除)，可擦除个别单元。写操作比读操作花更多的时间。集成度比 EPROM 低，而且更贵。

闪存(Flash Memory)

是一种新型的非易失性存储器。不像 RAM 那样需要电源支持才能保存信息，但又像 RAM 一样具有可写性。在某种低电压下，其内容可读不可写，此时类似于 ROM；在一种高

电压下，信息可更改或删除，这时又类似于RAM。常用于存储主板BIOS程序，或用作数码照相机存储卡和优盘，也可做成固态硬盘以代替磁盘存储器作为辅助存储器使用。

双口RAM(Dual-port RAM)

并行存储结构中的一种。双口RAM利用的是空间并行技术，它为一个存储体提供两组独立的读写控制电路和两个读写端口，因而可以对两个数据进行并行的读写。

多模块存储器(Multi-module Memory)

多模块存储器中包含多个小体，但每个体有其自己的MAR、MDR和读写电路，可独立组成一个存储模块。根据不同的编址方式可分为连续编址和交叉编址。

低位交叉编址(Low-order Interleaving)

也称交叉编址，低位表示存储器模块号，高位表示存储器模块内地址，使地址交叉分散在各模块内。

高位交叉编址(High-order Interleaving)

也称连续编址，主存地址高位表示存储器模块号，低位表示存储器模块内地址，使地址在同模块内连续编址。

高速缓存(cache)

在CPU和主存之间的一个高速小容量的存储器，在访问主存前先到该存储器访问。如果将当前正在访问的那个存储单元所在的主存块放到该存储器中，根据程序访问的局部化特性，这个主存块中的信息应该是最近经常要访问的，因此不必再到主存去访问，这样就可很快得到所需要的信息。

程序访问的局部性

对大量程序调查发现，程序在执行过程中产生的访存要求，其地址具有局部化特性。也就是说，在一个很小的时间段内，访问的存储器地址大多在一个局部区域内。体现在时间和空间两个方面，可分为时间局部性和空间局部性两种。

时间局部性(Temporal Locality)

时间局部性是指刚刚被访问的单元很可能在一个很短的时间内被再次访问。

空间局部性(Spatial Locality)

空间局部性是指刚刚被访问的单元的临近单元很可能不久也会被访问。

命中率(Hit Rate)

在快速的缓存中得到信息的概率。例如，在总共100次访问中，能在cache中访问到信息的次数为99次，则命中率为99%。

命中时间(Hit Time)

在命中情况下的访问时间。包括判断是否命中的时间和在上层快速存储器中的访问时间两部分。

缺失率(Miss Rate)

有些中文书翻译成“失靶率”或“失效率”。就是指没有命中的概率。如在总共100次访问中，能在cache中访问到信息的次数为99次，则缺失率为1%。

缺失损失(Miss Penalty)

指在缺失情况下，从主存取一个主存块到cache的时间，也称为不命中开销。

主存块(Block)

主存和 cache 之间进行信息交换的单位。把主存分成大小相等的块，主存块从 0 开始编号。访问某个主存单元时，就把这个单元所在的一个主存块调到 cache，根据程序访问的局部性特点，在随后的一段时间内，CPU 很可能要经常访问这个主存块。因为该主存块已调到 cache，所以很多时候就不需要访问主存了。

cache 槽或 cache 行(Slot/Line)

cache 由若干行组成，每一行中有一个用于存放主存块的槽，其大小与主存块一样，cache 行也从 0 开始编号，cache 行号就是槽号。

直接映射 cache(Direct-mapped cache)

把主存的每一块映射到 cache 的一个固定行中。这样，“主存块号”和“cache 行号”存在模映射关系，因此也称为模映射(Module Mapping)，即

cache 行(槽)号＝主存块号 mod cache 行数

全相联映射 cache(Fully Associative cache)

每个主存块可装入到 cache 任意一行中。每个 cache 行的标志字段指出了该行的数据信息取自主存的哪个块。

组相联映射 cache(Set-associative cache)

结合直接映射和全相联映射的特点，将 cache 所有行分组，把一个主存块映射到特定 cache 组的任一行中，即组间模映射、组内全映射。其映射关系为 cache 组号＝主存块号 mod cache 组数。

多级 cache(Multilevel cache)

在计算机系统中，同时使用多个层次的 cache。例如，在 CPU 和主存之间设置两级 cache：L1 cache 和 L2 cache。一般 L1 cache 是数据 cache 和代码 cache 分离的。

数据 cache(Data cache)

专门用来存放数据信息的高速缓冲存储器。

代码 cache(Code cache)

专门用来存放指令代码的高速缓冲存储器，也称为指令 cache。

分离式 cache(Split cache)

指数据和指令分开存放在各自的数据 cache 和指令 cache 中。

先进先出(First-In-First-Out，FIFO)

是一种替换算法，其基本思想是，总是把最先调入 cache 的一个主存块替换出去。

最近最少用(Least Recently Used，LRU)

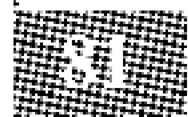
是一种替换算法，其基本思想是，总是把最近最少用的一个主存块从 cache 中替换出去。

全写(Write Through)

每次写 cache 的同时也写主存，主存与 cache 始终保持一致。这种方式比较简单，能保持主存与 cache 副本的一致性，但要插入慢速的访存操作，而且有些写入过程有可能是不必要的，例如中间结果的写入操作。这种方式的中文说法较多，有全写、直写、写直达、通过式写等。

写缓冲(Write Buffer)

在使用全写方式处理写操作时，为了减少每次写主存的时间，在 cache 和主存之间加一



个写缓冲。这样，不必每次都写主存，而只要写到一个快速的写缓冲就行了。当进行写操作时，CPU 先同时写 cache 和写缓冲，然后由存储控制器将写缓冲内容写到主存。

回写(Write Back)

每次写操作时，CPU 先暂时只写 cache，并用修改位(dirty bit)指明对应行中的信息是否被更新过。当某块内容需从 cache 中替换出去时，若信息被修改过，则将其一次写入主存，否则不需写主存。这种方式不在写 cache 中插入慢速的写主存操作，可以保持程序运行的快速性。但在写回主存前，主存与 cache 的内容可能不一致，因而会引起主存内容失效。这种方式的中文说法较多，有写回、回写、一次性写等。

3-C 模型(Three C's Model)

存储器层次结构中信息访问时的缺失类型以及结构的改变对这些缺失类型的影响，可以用一个模型来进行分析。该模型给出的 3 个缺失类型(强制缺失 Compulsory Miss、容量缺失 Capacity Miss、冲突缺失 Conflict Miss)名称的首字母正好都是“C”，所以称为 3-C 模型。

强制缺失(Compulsory Miss)

调入 cache 前第一次被用到时，会发生缺失。此时的缺失也被称为冷启动缺失(Cold-start Miss)。

容量缺失(Capacity Miss)

由于 cache 容量的限制，使有些主存块无法继续保存在 cache 中而造成缺失。其直接原因就是刚被替换出去的块很快又需要取回来。

冲突缺失(Conflict Miss)

这种缺失现象发生在组相联或直接映射 cache 中多个块同时竞争同一个位置时。

虚拟存储器(Virtual Memory)

虚拟存储器是一种存储管理机制，在采用虚拟存储器的系统中，每个作业运行时，可以只装入当前执行到的一部分到内存，而让暂时执行不到的另一部分放在磁盘上，当需要用到时再从磁盘装入到主存，这样使得在很小的主存空间能运行一个比它大的作业，而且用户编写程序时用到的逻辑地址空间可以比主存地址空间大。对用户来说，好像计算机系统具有一个容量很大的存储器，称为“虚拟存储器”。

物理存储器(Physical Memory)

通常把主存储器称为物理存储器。

虚拟地址(Virtual Address)

在虚拟存储管理机制中，每个源程序经编译、汇编、链接等处理生成可执行的二进制机器目标代码时，每个程序的目标代码都被映射到同样的虚拟地址空间。因此，通常把用户程序的指令及其操作数所在地址称为虚拟地址，或称为逻辑地址。虚拟地址的位数确定了虚拟地址空间的大小。例如，如果虚拟地址为 32 位，则虚拟地址空间大小为 2^{32}。

虚页号(Virtual Page Number，VPN)

为了实现分页虚拟存储管理机制，通常把虚拟地址空间划分为若干等长的块，每块称为一页(Page)。每页按顺序进行编号，从第 0 页开始，虚拟地址空间所包含的页数决定了虚页号的位数，虚拟地址的高位部分为虚页号。

页内偏移量(Page Offset)

页内偏移量指出需访问的逻辑地址位于当前页的哪个位置。页面大小决定了页内偏移量的位数,例如,如果一个虚拟页的大小为2K字节,那么,页内偏移量就是11位。它为虚拟地址的低位部分。

物理地址(Physical Address)

通常将主存储器地址称为物理地址,也称主存地址或实地址。

页框(Page Frame)

有些书把页框(有时也翻译为页帧)称为物理页或实页。操作系统在管理内存时,按页为单位进行内存分配。其具体做法是,把主存储器分成固定长且比较小的存储块,称为页框,每个进程也被划分成等长的程序块。这样,对进程进行存储分配时,将一个程序块(即虚拟页)装到一个可用的存储块(页框)中。

物理页号(Physical Page Number,PPN)

把主存空间分成固定长的页框,从0开始按顺序编号,该编号就是物理页号,也称为页框号或实页号。物理地址的高位部分是物理页号。

地址变换(Address Translation/Memory Mapping)

把指令中的虚拟地址转换为物理地址的过程称为地址变换。

重定位(Relocation)

在采用虚拟存储管理机制的系统中,每个进程都有一个同样的虚拟地址空间。在程序装入系统运行时,操作系统把用户程序的一部分或全部放到内存中,并把存放的物理地址信息记录到段表或页表中,以建立虚拟地址空间和物理地址空间之间的映射。实现这种映射的过程称为程序重定位,它建立了逻辑地址和物理地址的映射关系,实现了逻辑地址向物理地址的转换。所以,某种程度上重定位和地址转换是同一个概念。

有两种重定位方式:一种方式通过链接程序或加载程序进行地址转换而实现程序重定位,这种方式下,程序执行时每条指令中的地址已经是物理地址,称为静态重定位;另一种方式是在程序执行过程中由硬件动态实现地址转换,称为动态重定位。

页表(Page Table)

每个进程有一个页表,记录该进程的每个虚拟页存放在主存的哪个页框中,或在辅存哪个地方。页表中一般有装入位、修改位、替换控制位、访问控制位、物理页号等。

页表基址寄存器(Page Table Base Register)

每个进程有一个页表,页表在主存中的首地址被记录在一个特殊的寄存器中,这个特殊寄存器被称为页表基址寄存器,简称页表寄存器。

有效位(Valid Bit)

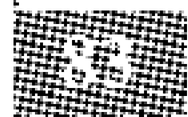
用来表示对应的虚页是否装入主存并有效,也称为装入位。若该位为"1",表示该页在主存中并且没有被淘汰。若该位为"0",则说明该页不在主存,发生了"缺页"异常。

修改位(Modify Bit)

用来表示对应的虚页在主存期间是否被修改过。若该位为"1",则表明该页已被修改过,淘汰时必须将该页写回到磁盘。若该位为"0",则表明该页未被修改过,淘汰时不需要将该页写回到磁盘。有些作者或系统把它称为"脏位(Dirty Bit)"。

使用位(Reference Bit/Use Bit)

用来表示对应虚页的使用情况,据此操作系统可以了解该页是最近经常被访问还是很



少被访问，因而确定该页是否马上被替换。所以，也称为替换控制位。

访问方式位(Access Bit)

用来表示虚页的读写权限。例如，代码段所在虚页的访问方式一般是“执行/只读”；共享数据段所在虚页一般是“只读”；私有数据段所在虚页一般是“可读可写”。该位也被称为访问控制位或存取权限位。

缺页(Page Fault)

需要访问的虚页不在内存中时，则发生“缺页”异常。计算机硬件通过检查页表项中的“有效位”就可以判断是否“缺页”。

交换(Swapping)/**页面调度**(Paging)

缺页时，需要把所缺页面从磁盘调到主存中，这个过程称为“页面换入”(或“磁盘调入”)；当需要从主存淘汰一页到磁盘时，称为“页面换出”(或“调出磁盘”)。页面换入/换出称为交换(Swapping)或页面调度(Paging)。

按需调度页面(Demand Paging)

只有发生“缺页”时才换入页面。大部分现代计算机系统都使用这种策略进行存储管理。

LRU 页(Least Recently Used Page)

指最近最少使用的页。通过检查“使用位”可以找到 LRU 页。

快表(Translation Lookaside Buffer，TLB)

用一个特殊的 cache 来跟踪记录最近用过的页表表项。因为页表表项主要用于地址转换，所以把这种特殊的 cache 称为转换后援缓冲器(Translation Lookaside Buffer，TLB)。因为在 TLB 中查找页表项速度很快，所以也称 TLB 为快表。TLB 通常很小，在高端机器中也通常不超过 128～256 项，一般用全相联方式，中等性能机器多用小的组相联方式。

分页式虚拟存储器(Paging VM)

分页式虚拟存储器的主要思想是，把主存储器分成固定长且比较小的存储块(称为页框，Page Frame)，虚拟地址空间也被划分成等长的程序块(称为页，Page)。操作系统把当前用到的页装入空闲的主存存储块中。所以分页方式是按固定长的页进行分配和调度的。逻辑地址由页号和页内偏移量组成。

分段式虚拟存储器(Segmentation VM)

分段式虚拟存储器与分页式虚拟存储器不同。分页式使用固定大小的块进行管理，而分段方式采用变长块的机制管理存储器。“段”是按照程序的逻辑结构划分而成的多个相对独立的部分。例如，过程、子程序、数据表、阵列等。操作系统在进行虚拟空间和主存空间对应时，按程序中实际的段来分配主存空间，每个段在主存中的起始位置记录在段表中，并附以“段长”项。段表本身也是主存中的一个可再定位段。一个大程序由多个代码段和多个数据段构成。逻辑地址由段号和段内地址组成。

段页式虚拟存储器(Paged Segmentation VM)

分段和分页相结合的方式。程序按独立模块分段，段内再分成固定大小的页，主存分配时仍以页为基本单位。用段表和页表(每段一个)进行两级定位管理。逻辑地址由段地址、页地址和偏移量 3 个字段构成。根据段地址到段表中查阅与该段相应的页表指针，转向页表，然后根据页地址从页表中查到该页在主存中的实页号，与偏移量相加得到物理地址。

进程(Process)

进程是程序在系统中某个数据集合上的一次动态运行。当你运行一个程序,就启动了一个进程,同一个程序在不同的数据集上运行构成不同的进程。进程是操作系统进行资源分配的单位。当某一个进程占用 CPU 执行,则该进程是活动进程(Active Process),否则是非活动进程 (Inactive Process)。操作系统通过把进程的状态装入相应的状态单元来使某个进程被激活。

管理模式(Supervisor Mode)

凡是用于完成操作系统各种功能的进程就是系统进程,也称为内核(Kernel)进程、管理(Supervisor)进程。此时,处理器所处的模式称为管理模式(Supervisor Mode),或称管理程序状态,简称管态、管理态、核心态。

用户模式(User Mode)

非操作系统功能的进程称为用户进程,当系统运行用户进程时,处理器的模式就是用户模式,或称用户状态、目标程序状态,简称为目态。

系统调用(System Call)

系统调用使 CPU 从用户态转换到管理态。系统调用指令是一种特殊的指令,执行这种指令后,CPU 就调出特定的操作系统内核模块进行执行,进入管理态。在管态下,操作系统可以执行专门的管态指令(或称特权指令,用户程序不能使用这些指令)来对一些用户进程不能访问的系统状态位或控制位进行读写。

异常返回(Return From Exception)

异常返回指令用于从操作系统内核进程返回到用户进程。通过系统调用进入操作系统内核时,需保留系统调用指令后面一条指令的地址,所以用异常返回指令返回用户进程时,可以根据该地址进行返回。

存储保护(Protection)

采用虚拟存储器的系统中,可以实现多道程序运行,也就是说,在一个主存物理空间中同时有多个进程共存。为避免主存中多道程序相互干扰,防止某进程出错而破坏其他进程的正确性,或某进程不合法地访问其他进程的存储区,应对每个进程进行存储保护。存储保护包含两个方面:(1)地址越界,即访问了不该访问的区域;(2)访问越权,即进行了不该进行的存取操作。

4.4 常见问题解答

1. ROM 和 RAM 一样,都是随机存取存储器吗?

答:是的。虽然经常把只读存储器 ROM 和随机访问存储器 RAM 放在一起进行分类,但 ROM 的存取方式和 RAM 是一样的,都是通过对地址进行译码后选择某个单元进行读写。所以两者采用的都是随机存取方式。不同的是,ROM 是只读的,RAM 是可读可写的。在程序执行过程中,ROM 存储区只能读出信息,不能修改,而 RAM 区可以读出,也可以修改信息。

2. 寄存器和主存储器都是用来存放信息的,它们有什么不同?

答:寄存器在 CPU 中,用触发器来实现,速度极快,价格高,容量只有几十个,多的机器

也通常只有几百个或几千个，主要用来暂存指令运行时的操作数和结果。

主存储器在 CPU 之外，用 MOS 管电路实现，速度没有寄存器快，价格也比寄存器便宜，容量可以达到 GB 数量级，用来存放被启动执行的程序代码及其数据。

3. 存取时间 Ta 就是存储周期 Tm 吗?

答：不是。存取时间 Ta 是执行一次读操作或写操作的时间，分为读出时间和写入时间。读出时间为从主存接收到有效地址开始到数据取出有效为止的最短时间；写入时间是从主存接收到有效地址开始到数据写入被写单元为止的最短时间。存储周期 Tm 是指存储器进行连续两次独立的读或写操作所需要的最小时间间隔。所以存取时间 Ta 不等于存储周期 Tm。通常存储周期 Tm 大于存取时间 Ta。

4. 刷新和再生是一回事吗?

答：不是一回事。对某个单元的刷新和再生的操作过程是一样的，即读后恢复，但再生操作是随机的，只对所读单元进行，而刷新操作则是按顺序定时一行一行进行的。

5. 刷新是一个个芯片按顺序完成的吗?

答：不是。刷新按行进行，每一行中的记忆单元同时被刷新，因此仅需要行地址，不需要列地址。刷新行号由 DRAM 芯片的刷新控制电路中的刷新计数器产生。存储器中的所有芯片的相同行同时进行刷新，例如，若有 8 个 1024×1024×4 的 DRAM 芯片构成一个存储器，则只需要 1024 次刷新操作就可以把整个存储器刷新一遍。

6. 主存都是由 RAM 组成的吗?

答：不是。主存是由 RAM 和 ROM 两部分组成的，它们统一编址，分别占用不同的地址范围。

7. 程序员是否需要知道高速缓存(cache)的访问过程?

答：不需要。cache 的访问过程对程序员来说是透明的。执行到一条指令时，需要到内存取指令，有些指令还要访问内存读取操作数或存放运算结果。采用 cache 的计算机系统中，总是先到 cache 去访问指令或数据，没有找到才到主存去访问。这个过程是 CPU 在执行指令过程中自动完成的。程序员不需要知道要找的指令和数据在不在 cache 中、在 cache 的哪一行，也不需要知道 cache 的访问过程，只要在指令中指定存储单元地址就行了。事实上，现代计算机都采用虚拟存储器机制，所以，在程序员编写的程序或编译链接生成的程序中，指令给出的地址还不是真正的内存单元地址，而是一个虚拟地址或逻辑地址，操作系统或硬件需要对程序进行重定位，以“页”或“段”为单位把程序装载到内存中，并把逻辑地址转换为真正的内存地址(物理地址)。

8. 主存和 cache 之间分块传送数据时，是否主存块越大，命中率越高?

答：不是。主存块大可充分利用程序访问的空间局部性特点，使得一个比较大的局部空间被一起调到 cache 中，因而可以增加命中机会。但是，主存块不能太大，主要原因有两个：(1)块大使得缺失损失变大，因为需花费更多时间从主存读一个较大的块。(2)块大则 cache 行数变少，因而替换可能性增加，导致命中的可能性变小。

9. 指令和数据都是放在同一个 cache 中的吗?

答：现代计算机系统中，一般采用多级 cache 系统。CPU 执行指令时，先到速度最快的一级 cache(L1 cache) 中寻找指令或数据，找不到时，再到速度次快的二级 cache(L2 cache) 中找，以此类推，最后到主存中找。对于一级 cache，指令和数据一般分开存放，而二级 cache

的指令和数据是放在一起的。因此，有 L1 data cache 和 L1 code cache。

10. cache 可以装在 CPU 芯片中吗?

答：可以。早期的计算机，其 cache 是装在主板上的。但随着 CPU 芯片技术的提高，cache 可以装在 CPU 中。从逻辑上来说，cache 是位于 CPU 和主存之间的部件，但在物理上，cache 被封装在 CPU 芯片内。目前，一级 cache、二级 cache，甚至三级 cache 都可以封装在 CPU 芯片中。

11. 直接映射方式下是否需要考虑替换策略？为什么?

答：无须考虑。因为在直接映射方式下，一个给定的主存块只能映射到一个固定的 cache 行中，所以，在对应 cache 行中已有一个主存块的情况下，新的主存块毫无选择地把原先已有的那个主存块替换掉，因而无须考虑替换算法。

12. 在 CPU 和主存之间加入了多个 cache，计算机总存储量就增加了，对吗?

答：不对。虽然 cache 是存储器，具有几百 KB 甚至几 MB 的容量，但因为它存放的是主存信息的副本，所以，并不能增加系统的存储容量。

13. 怎样保证 CPU 要找的指令和数据大都能在 cache 中访问到呢?

答：根据程序访问的局部性特点可知，不管是访问指令还是数据，CPU 在执行程序的过程中，若某个地址在 T 时刻被访问，则该地址及其邻近地址在 $T+\Delta t$ 时间段内很可能也被访问。因而，在访问到每个内存地址时，把该地址及其邻近的内存单元内容（即一个主存块）一起复制到 cache 中。这样，在接下来的一段时间内，CPU 所要访问的指令或数据基本上能在 cache 中找到了。

14. CPU 要找的指令和数据都能在 cache 中访问到吗？为什么?

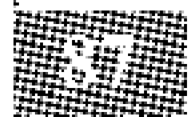
答：不能。指令/数据在第一次被访问时，肯定不在 cache 中，因而在 cache 中访问不到。此时，就会把所访问的指令/数据所在的主存块从主存取到 cache 中，这样，只要这个主存块不被其他主存块替换，以后再访问这个数据/指令或者同一块中其他数据/指令时，就能在 cache 中命中了。但是，随着程序的执行，CPU 所访问的地址区域会移到另外的主存块。由于 cache 容量的限制，当新的主存块调入 cache 时，原来在 cache 中的主存块很可能被新的主存块替换出来。如果替换出来后，CPU 又要对其进行访问，那么，CPU 在 cache 中肯定找不到。所以，CPU 要找的指令和数据不可能总在 cache 中访问到。

15. 发生取指令缺失时的处理步骤是什么?

答：每条指令执行的第一步是取指令。若在 cache 中取当前指令时发生缺失，则处理器必须按如下步骤完成：(1)把程序计数器的内容恢复为当前指令的地址，并通过地址线送主存储器。(2)控制主存储器执行一次读操作（若一个主存块只有一条指令，则一次读操作读一条指令即可；若一个主存块占用多条指令，则控制一次读出多条指令或读若干次），对主存的访问要通过总线完成，一次总线事务完成一次读操作。(3)读出的指令写到 cache 中，并把主存地址的高位写入到 cache 行的标记字段，最后设置有效位。(4)重新执行当前指令的第一步操作，即取指令，这次在 cache 中取指令时便能命中。

16. 引入 cache 后，CPU 的数据通路和控制部件要增加哪些功能和相关的电路?

答：在指令执行过程中，CPU 必须从存储器取指令，有些指令还要从存储器取操作数，或把结果写到存储器中。CPU 总是先到 L1 cache 中寻找，找不到再从 L2 cache 或主存中找。所以，在指令执行过程中，控制部件必须能够检测访问有没有命中。实现这个功能只要



用若干比较器和一些门电路即可。若命中，则直接在cache中访问即可。若缺失，则处理器需要进行一系列的处理。包括：使当前指令暂时停止执行，并冻结所有寄存器，然后用一个专门的控制器控制从下一级cache或主存中将当前访问地址所在的一个主存块送到cache中，并设置有效位和标志(Tag)信息，最后重新从暂停执行的时钟周期开始执行指令。

17. 写操作处理和读操作处理有什么不同？

答：因为读操作不改变cache中的信息，所以，读操作时的缺失处理比较简单。只要把主存块从主存装入cache中即可。而写操作时会改变cache中的信息，造成cache数据和主存数据的不一致。因此，要有相应的写策略来解决这种“不一致性”。

18. cache缺失对指令的执行有影响吗？有怎样的影响？

答：cache缺失对指令的执行效率有很大影响，会大大延长指令的执行时间。延长多少由缺失损失决定。若从L2 cache取，一般需要5～10个时钟周期，若从主存取，需要25～100个时钟周期。执行一条指令时的缺失情况有以下几种可能：(1)取指令时缺失，此时，从L2 cache或主存取出指令或指令所在的一个主存块放到cache后，再从头开始重新执行指令；(2)读数据时缺失，此时，从L2 cache或主存取出数据或数据所在的一个主存块到cache后，从取数那个时钟周期开始执行指令；(3)写数据时缺失，此时，需要根据相应的写策略来决定是当时就更新主存的数据还是在主存块被替换时更新主存的数据。存数操作结束后指令也就执行完了。

如果一条指令执行过程中，既发生取指令缺失又发生取数或存数缺失，那么，这条指令的执行将要延长多个缺失损失的时间。也就是说，可能会延长几十到几百个时钟周期。在流水线方式下，会大大影响指令的执行。

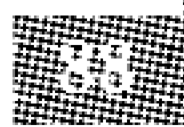
19. 虚拟存储器的大小是否等于磁盘的容量加上内存的容量？

答：不是。虚拟存储器本身只是一个概念，是一种存储管理机制，使用这种机制使得程序员编写程序时，好像计算机内部有一个极大的存储器，程序在这个极大的存储器中运行，而不受内存大小的限制。实际上这个存储器在物理上是不存在的，因此称为“虚拟”存储器。虚拟存储器的大小就是虚拟(或逻辑)地址空间的大小，它由逻辑地址的位数决定，与系统中所安装的磁盘容量和内存容量没有直接的关系。

20. 在存储器层次结构中，“cache—主存”、“主存—辅存”这两个层次有何异同点？

答：这两个层次在以下几个方面有相同之处：(1)都是基于程序访问的局部性特点，把连续的一块局部信息从慢速存储器复制到快速存储器；(2)都必须考虑慢速存储器和快速存储器之间的映射问题；(3)当需要在快速存储器中装入新的块而对应位置已满时，都需要考虑把哪一块从快速存储器中替换出来；(4)当在快速存储器中找不到信息时，都要从慢速存储器中将该信息所在块装入快速存储器中。

因为这两个层次所处的位置和引入的目的不同，所以它们之间也存在许多不同之处：(1)位置不同。cache最靠近CPU，辅存最远离CPU，CPU可以直接访问cache和主存，但不能直接访问辅存，辅存和主存直接交换数据。(2)目的不同。在CPU和主存之间加入cache，目的是为了加快CPU访问信息的速度；而在主存—辅存层次采用虚拟存储器机制是为了使程序员写程序时不受内存容量的限制。即扩大系统的存储容量。(3)交换的信息块大小不同。在“cache—主存”层次，交换的信息块称为“主存块(Block)”，一般大小为8～128字节；而在“主存—辅存”层次，交换的信息块称为“虚拟页(Page)”，一般大小为4K～64K字



节。随着技术的发展，块大小和页大小都可能变化，但它们之间在数量级上的差别总是存在的，而且会很大。因为虚拟页缺失损失比 cache 缺失损失大得多，所以虚拟页太小会影响命中率从而极大降低系统效率。(4)缺失处理不同。在“cache—主存”层次，缺失处理由处理器硬件来实现；而在“主存—辅存”层次，则由操作系统软件来实现。(5)映射方式不同。在“cache—主存”层次，可根据不同的情况选择使用直接、全相联或组相联方式，映射关系完全由硬件来实现；而在“主存—辅存”层次，则都采用全相联方式，映射关系由操作系统查询页表来实现。(6)写策略不同。在“cache—主存”层次，可以采用“全写”和“回写”两种策略；但在“主存—辅存”层次，则都采用“回写”策略。因为，如果采用“全写”，每次写操作都要访问磁盘，这样的开销是不能容忍的。

21. 所有程序都具有同样的虚拟地址空间，会不会发生信息被互相读写的情况呢？

答：不会。虚拟存储机制使得每个程序员在一个很大的虚拟地址空间中编写程序，不必考虑主存有多大，也不必考虑其他程序用的地址是否和自己用的地址有冲突。也就是说，每个程序都具有同样的虚拟地址空间。用户程序加载后，操作系统内核会为该程序生成一个进程。在进程执行过程中，操作系统会按照某种存储管理机制(分段、分页、段页)把当前需要的用户程序的一部分从磁盘调到主存中，并把所在的物理地址信息记录到段表或页表中，进行虚拟地址空间到物理地址空间的映射。因此，CPU 访问信息的真正地址是主存物理地址。虽然两个用户程序中用到的逻辑地址是一样的，但由于两个用户程序被存放到不同的物理主存区，因而，不会发生信息被互相读写的问题，即使由于程序错误而导致这种问题也很容易发现。

22. 装入一个新的页面时，主存没有空闲页框，怎么办？

答：装入一个新页时，需要到主存找一个空闲页框。如果主存没有空闲页框，则必须选择一个虚拟页面从主存的某个页框中替换出来。

23. 怎么知道要找的页面不在内存？

答：所谓要找的页面不在内存，实际上就是在取某条指令或存取某个操作数时，发生了“缺页”情况。是否“缺页”主要是通过查看对应页表项中的“有效位”是否为“0”来判断。大致过程如下：根据要找的指令或操作数的地址高位，确定所访问的虚页号，以虚页号作为索引值，找到对应的页表项，每个页表项中都有一个“有效位”，若为“0”表示该虚页(即指令所在的程序块或操作数所在的数据块)不在内存，发生了“缺页”异常。

24. 每次进行存储访问时，总是先要进行逻辑地址到物理地址的转换吗？

答：如果采用的是动态重定位，则执行指令过程中只要进行存储访问，总是先要进行逻辑地址到物理地址的转换。如果采用的是静态重定位，则指令中的地址已经是物理地址，故存储访问时不需要地址转换。

25. 逻辑地址到物理地址的转换是由硬件实现的还是软件实现的？

答：动态重定位方式下，由专门的硬件(存储器管理部件 MMU)实现逻辑地址到物理地址的转换。静态重定位方式下，由软件(链接程序或加载程序)实现地址转换。

26. 快表(TLB)在主存还是在高速缓存中？

答：为了尽量避免到主存访问页表，通常把最近经常访问的页表项放到一个特殊的高速缓存中，这个存放若干页表项的特殊 cache 称为快表 TLB。所以，快表在高速缓存中。

27. CPU 执行指令进行一次存储访问操作要访问主存几次？

答：在具有 cache 并采用动态重定位存储管理的系统中，一次存储访问的大致过程如下：

(1) 根据虚页号查快表，若快表中有对应虚页的页表项，则取出页框号形成物理地址，转(2)；若快表中不存在对应虚页的页表项，则发生 TLB 缺失，转(3)。

(2) 判断物理地址中的标记是否和 cache 中标记相等并且有效位是否为“1”，是，则 cache 命中，从 cache 取数或写数据到 cache(“全写”方式下，同时也写主存)中；不是，则发生 cache 缺失，转(4)。

(3) 当 TLB 缺失时，根据页表基址寄存器的值和虚页号找到主存中的页表项，判断有效位是否为“1”，若是，则说明该虚页在主存中，此时，把该页表项装入 TLB 中，并取出页框号形成物理地址，转(2)；若不是，则说明该虚页不在主存中，即发生了“缺页”异常。此时，需要调出操作系统中的“缺页”异常处理程序，实现从磁盘读入一个页面的功能。“缺页”处理结束后，重新执行当前指令，这次一定能在主存中找到。

(4) cache 缺失时，CPU 根据物理地址到主存读一块信息到 cache，然后再读入到 CPU 或 CPU 写信息到 cache 中。

从上述过程来看，CPU 进行一次存储访问操作，最好的情况下无须访问主存，最坏的情况下，不仅要多次访问主存，还要读写磁盘数据。

28. 具有 TLB、cache 和虚拟存储管理机制的计算机系统中，有没有可能出现“cache 命中但缺页”的情况？那“TLB 命中但缺页”的情况有没有可能发生呢？

答：不可能出现“cache 命中但缺页”的情况。因为如果缺页，说明当前页面不在主存中，那么，也一定不在 cache 中。同样，“TLB 命中但缺页”的情况也不可能发生。因为若当前页面不在主存中，那么，TLB 中不可能有该页对应的页表项。

29. 快表缺失、cache 缺失和页面缺失(缺页)的处理有什么异同点？

答：cache 缺失的处理是由硬件实现的。当发生 cache 缺失时，CPU 使当前指令阻塞，并根据主存地址继续到主存中去访问主存块，从主存中取到信息后指令继续执行。

TLB 缺失可以用软件也可以用硬件来处理。首先根据虚页号和页表基址寄存器的内容到主存中找到相应的页表项，若有效位为“1”，则把该项取到 TLB 中即可。若有效位为“0”，则发出“缺页”异常。用软件实现时，通过产生一个“TLB 缺失异常”，调出操作系统中相应的异常处理程序，异常处理结束后，重新执行当前指令。

页面缺失(缺页)处理是由软件实现的。缺页时，需要调出操作系统中的“缺页”异常处理程序进行处理，实现从磁盘读入一个页面的功能。“缺页”处理结束后，重新执行当前指令。

30. 存储器分层结构中，各层次上的存储器的速度如何？

答：在计算机系统中，存储器采用的是一种分层结构，包括寄存器—cache—主存—磁盘。它们之间的相对速度若用一个 CPU 时钟周期来表示，则在 0 个周期内就能从寄存器访问到信息；在 1～10 个时钟周期能够在 cache 中访问到信息；在 50～100 个时钟周期能在主存中访问到信息；如果要从磁盘读信息，则大约需要几十万到几百万个时钟周期。因此，程序员必须能够充分理解存储器的分层结构，它对程序的性能有巨大的影响。随着技术的进步各种存储器的速度可能会发生变化，但它们之间的差异总是存在的。

4.5 单项选择题

1. 下面有关半导体存储器组织的叙述中，错误的是（　　）。
   A. 存储器的核心部分是存储体，由若干存储单元构成
   B. 存储单元由若干个存放 0 或 1 的存储元件构成
   C. 一个存储单元有一个编号，就是存储单元的地址
   D. 同一个存储器中，每个存储单元的宽度可以不同
2. 下面（　　）存储器是目前已被淘汰的存储器。
   A. 半导体存储器　　B. 磁表面存储器
   C. 磁芯存储器　　D. 光盘存储器
3. 若计算机的主存储器容量为 1GB，也就等于（　　）。
   A. 2^{30} 个字节　　B. 10^{30} 个字节
   C. 2^{9} 个字节　　D. 10^{9} 个字节
4. 若 SRAM 芯片的容量为 1024×4 位，则地址和数据引脚的数目分别为（　　）。
   A. 10,4　　B. 5,4　　C. 10,8　　D. 5,8
5. 若计算机字长 16 位，主存地址空间大小是 64KB，按字节编址，则主存寻址范围是（　　）。
   A. 0～(64K－1)　　B. 0～(32K－1)
   C. 0～(64KB－1)　　D. 0～(32KB－1)
6. EPROM 是指（　　）。
   A. 读写存储器　　B. 掩膜只读存储器
   C. 可编程的只读存储器　　D. 可擦除可编程的只读存储器
7. 下列几种存储器中，（　　）是易失性存储器。
   A. cache　　B. EPROM
   C. Flash Memory　　D. CD-ROM
8. 假定主存地址空间大小为 1024MB，按字节编址，每次读写操作最多可以一次存取 32 位。不考虑其他因素，则存储器地址寄存器 MAR 和存储器数据寄存器 MDR 的位数至少应分别为（　　）。
   A. 30,8　　B. 30,32　　C. 28,8　　D. 28,32
9. 需要定时刷新的半导体存储器芯片是（　　）。
   A. SRAM　　B. DRAM　　C. EPROM　　D. Flash Memory
10. 通常采用行、列地址引脚复用的半导体存储器芯片是（　　）。
    A. SRAM　　B. DRAM　　C. EPROM　　D. Flash Memory
11. 具有 $\overline{RAS}$（行地址选通）和 $\overline{CAS}$（列地址选通）信号引脚的半导体存储器芯片是（　　）。
    A. SRAM　　B. DRAM　　C. EPROM　　D. Flash Memory
12. 下面有关系统主存储器的叙述中，错误的是（　　）。
    A. RAM 是可读可写存储器，ROM 是只读存储器

B. ROM和RAM都采用随机访问方式进行读写

C. 系统的主存由RAM和ROM组成

D. 系统的主存都用DRAM芯片实现

13. 下面有关半导体存储器的叙述中，错误的是（　　）。

A. 半导体存储器都采用随机存取方式进行读写

B. ROM芯片属于半导体随机存储器芯片

C. SRAM是半导体静态随机访问存储器，可用作cache

D. DRAM是半导体动态随机访问存储器，可用作主存

14. 假定用若干个16K×1位的存储器芯片组成一个64K×8位的存储器，芯片内各单元连续编址，则地址BFF0H所在的芯片的最小地址为（　　）。

A. 4000H　　B. 6000H　　C. 8000H　　D. A000H

15. 假定用若干16K×8位的存储器芯片组成一个64K×8位的存储器，芯片各单元交叉编址，则地址BFFF所在的芯片的最小地址为（　　）。

A. 0000H　　B. 0001H　　C. 0002H　　D. 0003H

16. 用存储容量为16K×1位的存储器芯片组成一个64K×8位的存储器，则在字方向和位方向上分别扩展了（　　）倍。

A. 4和2　　B. 4和8　　C. 2和4　　D. 8和4

17. 存储容量为16K×4位的DRAM芯片，其地址引脚和数据引脚数各是（　　）。

A. 7和1　　B. 7和4　　C. 14和1　　D. 14和4

18. 多模块存储器之所以能被快速访问，是因为（　　）。

A. 采用了高速元器件　　B. 各模块有独立的读写电路

C. 采用了信息预读技术　　D. 模块内各单元地址不连续

19. 相联存储器是按（　　）进行寻址访问的存储器。

A. 地址指定方式　　B. 内容指定方式

C. 堆栈访问方式　　D. 队列访问方式

20. 在存储器分层体系结构中，存储器速度从最快到最慢的排列顺序是（　　）。

A. 寄存器—主存—cache—辅存　　B. 寄存器—主存—辅存—cache

C. 寄存器—cache—辅存—主存　　D. 寄存器—cache—主存—辅存

21. 在存储器分层体系结构中，存储器从容量最大到最小的排列顺序是（　　）。

A. 主存—辅存—cache—寄存器　　B. 辅存—cache—主存—寄存器

C. 辅存—主存—cache—寄存器　　D. 辅存—主存—寄存器—cache

22. 在主存和CPU之间增加cache的目的是（　　）。

A. 增加内存容量　　B. 提高内存可靠性

C. 加快信息访问速度　　D. 增加内存容量，同时加快访问速度

23. 以下哪一种情况能很好地发挥cache的作用？（　　）。

A. 程序中不含有过多的I/O操作

B. 程序的大小不超过实际的内存容量

C. 程序具有较好的访问局部性

D. 程序的指令间相关度不高

24. 假定主存地址位数为32位，按字节编址，主存和cache之间采用直接映射方式，主存块大小为1个字，每字32位，写操作时采用全写(Write Through)方式，则能存放32K字数据的cache的总容量至少应有多少位？(　　)。

A. 1504K　　B. 1536K　　C. 1568K　　D. 1600K

25. 假定主存地址位数为32位，按字节编址，主存和cache之间采用直接映射方式，主存块大小为1个字，每字32位，写操作时采用回写(Write Back)方式，则能存放32K字数据的cache的总容量至少应有多少位？(　　)。

A. 1504K　　B. 1536K　　C. 1568K　　D. 1600K

26. 假定主存地址位数为32位，按字节编址，主存和cache之间采用全相联映射方式，主存块大小为1个字，每字32位，采用回写(Write Back)方式和随机替换策略，则能存放32K字数据的cache的总容量至少应有多少位？(　　)。

A. 1536K　　B. 1568K　　C. 2016K　　D. 2048K

27. 假定主存按字节编址，cache共有64行，采用直接映射方式，主存块大小为32字节，所有编号都从0开始。问主存第3000号单元所在主存块对应的cache行号是(　　)。

A. 13　　B. 26　　C. 29　　D. 58

28. 假定主存按字节编址，cache共有64行，采用4路组相联映射方式，主存块大小为32字节，所有编号都从0开始。问主存第3000号单元所在主存块对应的cache组号是(　　)。

A. 1　　B. 5　　C. 13　　D. 29

29. 假定采用单体存储器组织方式，CPU通过存储器总线读取数据的过程为：发送地址和读命令需1个时钟周期，存储器准备一个数据需8个时钟周期，总线上每传送1个数据需1个时钟周期。若主存和cache之间交换的主存块大小为64B，存取宽度和总线宽度都为4B，则cache的一次缺失损失至少为多少个时钟周期？(　　)。

A. 64　　B. 72　　C. 80　　D. 160

30. 假定采用单体存储器组织方式，CPU通过存储器总线读取数据的过程为：发送地址和读命令需1个时钟周期，存储器准备一个数据需8个时钟周期，总线上每传送1个数据需1个时钟周期。若主存和cache之间交换的主存块大小为64B，存取宽度和总线宽度都为8B，则cache的一次缺失损失至少为多少个时钟周期？(　　)。

A. 64　　B. 72　　C. 80　　D. 160

31. 假定采用多模块交叉存储器组织方式，存储器芯片和总线支持突发传送，CPU通过存储器总线读取数据的过程为：发送首地址和读命令需1个时钟周期，存储器准备第一个数据需8个时钟周期(即CAS潜伏期＝8)，随后每个时钟周期总线上传送1个数据，可连续传送8个数据(即突发长度＝8)。若主存和cache之间交换的主存块大小为64B，存取宽度和总线宽度都为8B，则cache的一次缺失损失至少为多少个时钟周期？(　　)。

A. 17　　B. 20　　C. 33　　D. 65

32. 以下是有关虚拟存储管理机制中地址转换的叙述，其中错误的是(　　)。

A. 地址转换是指把逻辑地址转换为物理地址

B. 一般来说，逻辑地址比物理地址的位数少

C. 地址转换过程中会发现是否“缺页”

D. MMU在地址转换过程中要访问页表项

33. 下列命中组合情况中，一次访存过程中不可能发生的是（　　）。

A. TLB命中、cache命中、Page命中

B. TLB未命中、cache命中、Page命中

C. TLB未命中、cache未命中、Page命中

D. TLB未命中、cache命中、Page未命中

34. 以下是有关虚拟存储管理机制中页表的叙述，其中错误的是（　　）。

A. 系统中每个进程有一个页表

B. 页表中每个表项与一个虚拟页对应

C. 每个页表项中都包含装入位（有效位）

D. 所有进程都可以访问页表

35. 以下是有关“缺页”处理的叙述，其中错误的是（　　）。

A. 缺页处理过程中需要修改TLB

B. 缺页是一种外部中断，需要调用操作系统提供的中断服务程序来处理

C. 缺页处理过程中需根据页表中给出的磁盘地址去读磁盘数据

D. 缺页处理完后要重新执行发生缺页的指令

36. 以下是有关页式虚拟存储器的叙述，其中错误的是（　　）。

A. 进程被划分成等长的虚拟页，内存被划分成同样大小的页框

B. 采用全相联映射，每个虚拟页可以映射到任何一个空闲的页框中

C. 当从磁盘装入的信息不足一页时会产生页内碎片

D. 相对于段式虚拟存储器，分页方式更利于存储保护

37. 以下是有关段式虚拟存储器的叙述，其中错误的是（　　）。

A. 段是逻辑结构上相对独立的程序块，因此段是可变长的

B. 按程序中实际的段来分配主存，被分配后的主存块是可变长的

C. 每个段表项必须记录对应段在主存的起始位置和段的长度

D. 分段方式对低级语言程序员和编译器来说是透明的

38. 以下是有关快表的叙述，其中错误的是（　　）。

A. 快表的英文缩写是TLB，称为转换后援缓冲器

B. 快表中存放的是当前进程的常用页表项

C. 若在快表中命中，则在L1 cache中一定命中

D. 快表是一种高速缓存，通常集成在CPU芯片中

【参考答案】

1. D	2. C	3. A	4. A	5. A	6. D	7. A
8. B	9. B	10. B	11. B	12. D	13. A	14. C
15. D	16. B	17. B	18. B	19. B	20. D	21. C
22. C	23. C	24. B	25. C	26. D	27. C	28. C
29. D	30. C	31. A	32. B	33. D	34. D	35. B
36. D	37. D	38. C				

4.6 分析应用题

1. 假定某计算机的主存地址空间大小为512MB,按字节编址。若每次读写操作最多可以存取32位,则存储器地址寄存器MAR和存储器数据寄存器MDR的位数至少分别为多少?

【分析解答】

主存地址空间大小为512MB,按字节编址,说明每个存储单元有8位,共有$512M=2^{29}$个存储单元。因而,地址位数至少应有29位,故存放主存地址的存储器地址寄存器MAR至少应有29位。每次读写最多存取32位,因此,用来作为读/写数据缓冲的存储器数据寄存器MDR的位数至少应有32位。

2. 某计算机主存地址16位,每个存储单元有8位,即按字节编址。如果用1K×4位的RAM芯片构成该计算机的最大主存空间,需要多少芯片?片选逻辑的输入需要多少位地址?

【分析解答】

因为主存地址为16位,所以主存地址空间大小为64K个存储单元,每个存储单元占8位。因此需要的芯片数为(64K/1K)×(8/4)=64×2=128。存储器在字方向上扩展了64倍,因而片选逻辑需要6位地址。每个芯片有$1K=1024=2^{10}$个单元,因此芯片内地址位数为10位,剩下6位地址正好用于片选逻辑。

3. 构成256K×8位的存储器,需多少个64K×1位的DRAM芯片?存储器所有单元刷新一遍需要多少次刷新操作?若采用异步刷新方式,每个单元刷新间隔不超过2ms,则生成的刷新信号的间隔时间是多少?若采用集中刷新方式,则存储器刷新一遍最少用多少个读写周期?若改用16K×4位的DRAM芯片构成上述256K×8位的存储器,则存储器所有单元刷新一遍需要多少次刷新操作?

【分析解答】

该存储器所需芯片数为(256K/64K)×(8/1)=32。因为所用芯片为64K×1位的DRAM芯片,因而芯片中只有一个位平面,256×256的存储阵列结构构成了64K个单元,构成存储器的所有芯片同时按行刷新,每个芯片有256行,所以存储器所有单元刷新一遍至少需要256次刷新操作。若采用异步刷新方式,则相邻两次刷新信号产生的间隔时间为$2ms/256\approx 7.8\mu s$。若采用集中刷新方式,则整个存储器刷新一遍最少用256个读写周期,在这个过程中,存储器不能进行读写操作。

若改用16K×4位的DRAM芯片,则每个芯片中的存储阵列由4个128×128的位平面构成,则4个存储阵列中行号相同的那些行同时进行刷新操作,共有128行,因而整个存储器刷新一遍只需要128次刷新操作。

4. 某计算机的主存地址空间大小为64KB,按字节编址,已配有0000H～7FFFH的ROM区,若再用8K×4位的RAM芯片形成其余32K×8位的RAM存储区,则需要多少个这样的RAM芯片?假定将该计算机的主存地址空间升级为16MB,ROM区地址范围还是000000H～007FFFH,剩下的所有地址空间都用8K×4位的RAM芯片配置,则需要多少个这样的RAM芯片?

【分析解答】

因为主存地址空间为64KB，按字节编址，所以主存地址范围为0000H～FFFFH，其中0000H～7FFFH为ROM区，8000H～FFFFH为RAM区，当A_{15}为0时选中ROM芯片，当A_{15}为1时选中RAM芯片。因为RAM区的大小为32KB，故需8K×4位的RAM芯片数为32KB/(8K×4b)＝4×2＝8。若主存地址空间升级为16MB时，主存地址为24位，ROM区范围为000000H～007FFFH，其大小为32KB，主存空间总大小为16MB＝512×32KB，所以RAM区大小为511×32KB，需使用的RAM芯片数为511×32KB/(8K×4b)＝511×4×2＝4088。

5. 假设CPU有16根地址引脚，8根数据引脚，并用$\overline{MREQ}$作为访存控制信号(低电平有效)，用$\overline{WR}$作为读/写控制信号(低电平为写，高电平为读)。现有以下规格的存储器芯片：1K×4位RAM、4K×8位RAM、1K×8位ROM、4K×8位ROM，另外有74LS138译码器和各种门电路。主存地址空间分配为：6000H～67FFH为ROM区，6800H～6BFFH为RAM区，其余为备用区，暂不连接芯片。请画出CPU与存储器的连接图，并详细画出片选逻辑。

【分析解答】

第一步：将十六进制地址写成对应的二进制地址形式。

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	ROM区
							…									
0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	2K×8位
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	RAM区
							…									
0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1K×8位

第二步：选择芯片。ROM区选择2片1K×8位ROM芯片；RAM区选择2片1K×4位RAM芯片。选其他芯片都不合理。

第三步：地址线的连接。对于1K×8位的ROM芯片和1K×4位的RAM芯片来说，其芯片内的地址位数都为10位，因此，每个芯片的地址引脚都分别与地址线$A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$相连。剩下的高位地址线和访存控制信号$\overline{MREQ}$共同产生片选信号。

第四步：片选信号的形成。按本题要求，$A_{15}A_{14}=01$，ROM区的$A_{13}A_{12}A_{11}=100$，RAM区的$A_{13}A_{12}A_{11}A_{10}=1010$。74LS138译码器要求控制端$G_1$为高，$\overline{G_{2A}}$与$\overline{G_{2B}}$为低，因此可把它们分别接到$A_{14}$、$A_{15}$和$\overline{MREQ}$上。地址线$A_{13}A_{12}A_{11}$可作为译码器的C、B、A输入端。最终的片选信号由译码器输出信号和地址线A_{10}组合生成。具体的连接如图4.2所示，其中$\overline{CS}$为片选信号。

6. 假定一个存储器系统支持四体交叉存取，某程序执行过程中，CPU访问的主存地址序列为3,9,17,2,51,37,13,4,8,41,67,10，则哪些地址访问会发生体冲突？

【分析解答】

对于四体交叉访问的存储系统，理想情况下，每隔1/4周期可读写一个数据，假定这个

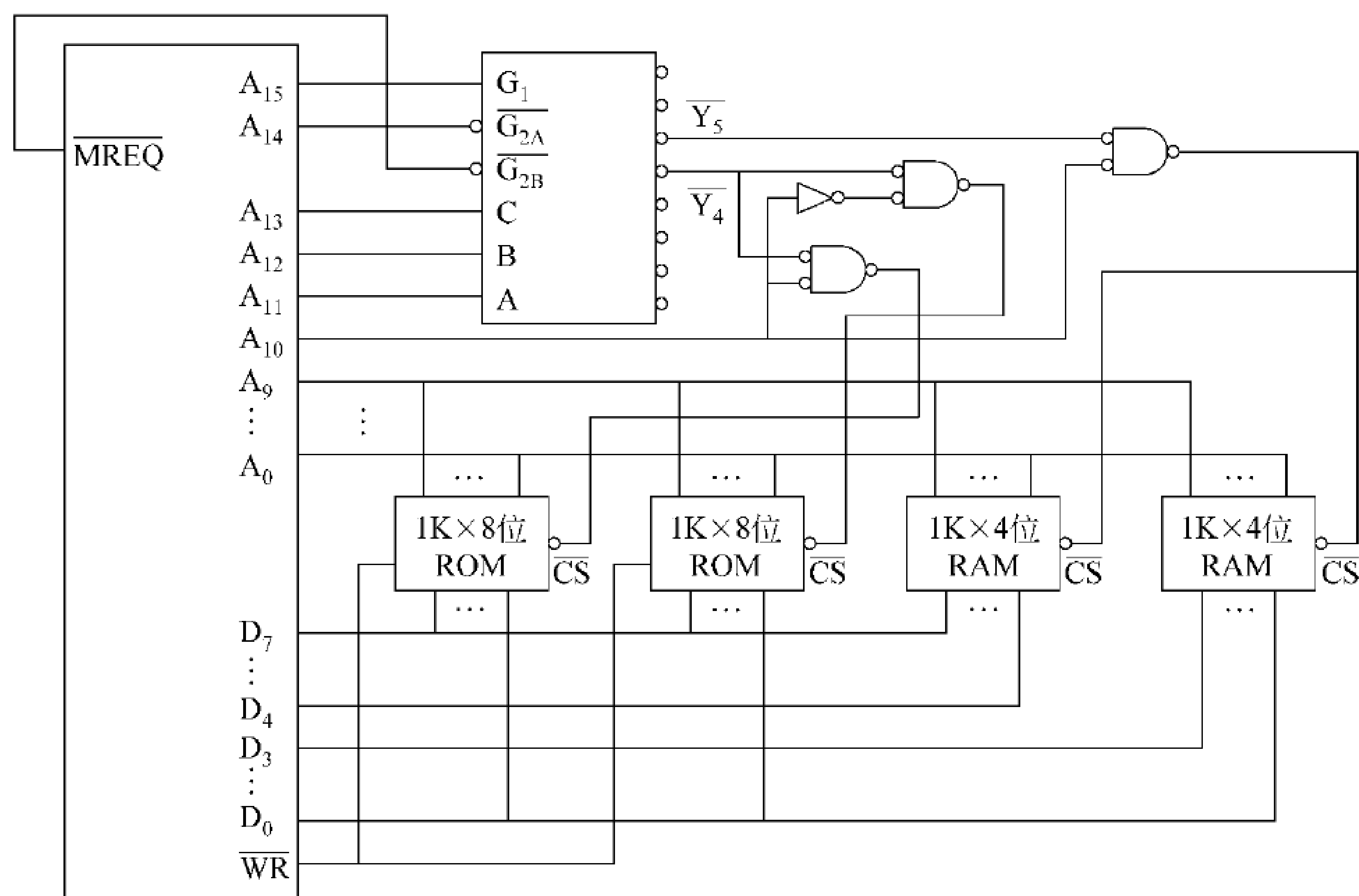


图 4.2 题 5 中 CPU 与存储芯片的连接

时间为 Δt。每个存储模块内的地址分布如下。

模块 0：0、4、8、12、16、…

模块 1：1、5、9、13、17、…、37、…、41、…

模块 2：2、6、10、14、18、…

模块 3：3、7、11、15、19、…、51、…、67、…

很显然，如果相邻四次访问中给定的访存地址出现在同一个模块内，就会发生访存冲突。所以 17 和 9、37 和 17、13 和 37、8 和 4 会发生冲突。41 和 13 也在同一个模块内且访问间隔小于 4 个 Δt，但是，由于访问第 8 单元发生冲突而使其访问延迟 3 个 Δt 进行，从而使得 41 号单元也延迟 3 个 Δt 访问，因而，其访问不会和 13 号单元的访问发生冲突。

7. 现代计算机中，SRAM 一般用于实现快速小容量的 cache，而 DRAM 用于实现慢速大容量的主存。以前超级计算机通常不提供 cache，而是用 SRAM 来实现主存(如 Cray 巨型机)，请问：如果不考虑成本，你还这样设计高性能计算机吗？为什么？

【分析解答】

不会。其理由主要有以下两个方面：

(1) 主存越大越好，主存大，缺页率降低，因而减少了访问磁盘所需要的时间。DRAM 芯片的集成度比 SRAM 芯片的高得多，因而，用 DRAM 芯片比用 SRAM 芯片构成的主存容量大得多。

(2) 程序访问的局部性特点使得 cache 的命中率很高，因而，CPU 访问的主要是 cache，对主存的访问不多，而且现代 DRAM 芯片中也有 SRAM 构成的高速缓存区。因此即使主存没有使用快速的 SRAM 芯片而是用 DRAM 芯片，也不会对存储访问速度有多大影响。

8. 某计算机主存地址空间大小有 8MB，分成 32768 个主存块，按字节编址；cache 可存

放 8KB 数据(不包括有效位、标记等附加信息),采用直接映射方式,问 cache 共有多少行?主存地址如何划分?要求说明每个字段的含义、位数和在主存地址中的位置。

【分析解答】

每个主存块大小为 8MB/32768=256B,故 cache 共有 8KB/256B=32 行。直接映射方式下,cache 行号(即行索引)有 5 位;由于每个主存块大小为 256B,按字节编址,故块内地址为 8 位;因为主存地址空间大小为 8MB,所以主存地址位数为 23 位,故主存地址中标记有 23-5-8=10 位。综上所述,主存地址共有以下 3 个字段:高 10 位为标记,中间 5 位为行索引,低 8 位为块内地址。

9. 某计算机主存地址空间大小为 1GB,按字节编址。cache 可存放 64KB 数据,主存块大小为 128 字节,采用直接映射和全写(Write Through)方式。请回答下列问题:

(1) 主存地址如何划分?要求说明每个字段的含义、位数和在主存地址中的位置。

(2) cache 的总容量为多少位?

【分析解答】

(1) cache 共有 64KB/128B=512 行,直接映射方式下,cache 行号占 9 位;由于每个主存块大小为 128B,按字节编址,故块内地址为 7 位;主存地址空间大小为 1GB,所以地址位数为 30 位。主存地址中标记有 30-9-7=14 位。综上所述,主存地址共有以下 3 个字段:高 14 位为标记,中间 9 位为行索引,低 7 位为块内地址。

(2) 因为直接映射不考虑替换算法,所以 cache 行中没有用于替换的控制位;因为采用全写方式,所以,cache 行中也没有修改位。每个 cache 行中包含 1 位有效位、14 位标记位和 128B 的数据,因此,cache 总容量为 $512\times(1+14+128\times8)$b=519.5Kb。

10. 对于数据的访问,分别给出具有下列要求的程序或程序段的示例。

(1) 空间局部性和时间局部性都好。

(2) 时间局部性好,空间局部性差。

(3) 空间局部性好,时间局部性差。

(4) 时间局部性和空间局部性都差。

【分析解答】

对于按行优先存放在内存的多维数组,如果按列优先访问数组元素,则空间局部性就差;如果在一个循环体中某个数组元素只被访问一次,则时间局部性就差。假定二维数组 a[1000][1000]按行优先存放在内存,以下给出的 4 个程序片段用于对数组 a 进行相应的处理,它们具有相同的功能,但数组访问的时间局部性和空间局部性截然不同(不考虑编译器的优化)。

(1) 时间局部性和空间局部性都好

```
for (i=0; i<1000; i++)
    for (j=0; j<1000; j++) {
        sum=sum+a[i][j];
        product=product* a[i][j];
        square=square+a[i][j]* a[i][j];
    }
```

(2) 时间局部性好,空间局部性差

```
for (j=0; j<1000; j++)
    for (i=0; i<1000; i++) {
        sum=sum+a[i][j];
        product=product* a[i][j];
        square=square+a[i][j]* a[i][j];
    }
```

(3) 空间局部性好,时间局部性差

```
for (i=0; i<1000; i++)
    for (j=0; j<1000; j++)
        sum=sum+a[i][j];
for (i=0; i<1000; i++)
    for (j=0; j<1000; j++)
        product=product* a[i][j];
for (i=0; i<1000; i++)
    for (j=0; j<1000; j++)
        square=square+a[i][j]* a[i][j];
```

(4) 时间局部性和空间局部性都差

```
for (j=0; j<1000; j++)
    for (i=0; i<1000; i++) {
        sum=sum+a[i][j];
for (j=0; j<1000; j++)
    for (i=0; i<1000; i++) {
        product=product* a[i][j];
for (j=0; j<1000; j++)
    for (i=0; i<1000; i++) {
        square=square+a[i][j]* a[i][j];
```

11. 假定某计算机的主存地址空间大小为64KB,按字节编址;cache采用4-路组相联映射、LRU替换和写回(Write Back)策略,能存放4KB数据,主存与cache之间交换的主存块的大小为64字节。请回答下列问题:

(1) 主存地址字段如何划分?要求说明每个字段的含义、位数和在主存地址中的位置。

(2) cache的总容量有多少位?

(3) 若cache初始为空,CPU依次从0号地址单元顺序访问到4344号单元,共重复访问16次。cache存取时间为20ns,主存存取时间为200ns,试估计CPU访存的平均时间。

【分析解答】

(1) cache的行数为4KB/64B=64;因为采用4-路组相联,所以每组有4行,共16组。主存地址空间大小为64KB,故主存地址有16位,其中低6位为块内地址,中间4位为组号(组索引),高6位为标记。

(2) 因为采用写回策略,所以cache每行中要有一个修改位(Dirty Bit);因为每组有4行,所以每行有两位LRU位。此外,每行还有6位标记、1位有效位和64字节数据,共有64

行，故总容量为 64×(6+1+1+2+64×8)=33408 位。

(3) 块大小为 64 字节，CPU 总共访问了 4345 个单元，4345/64=67.89，因此第 0～4344 单元应该对应前 68 块(第 0～67 块)，即 CPU 访问过程是对主存的前 68 块连续访问 16 次。图 4.3 给出了访问过程中主存块和 cache 行之间的映射关系。图中列方向是 cache 的 16 个组，行方向是每组的 4 行。

	第0行	第1行	第2行	第3行
第0组	0/64/48	16/0/64	32/16	48/32
第1组	1/65/49	17/1/65	33/17	49/33
第2组	2/66/50	18/2/66	34/18	50/34
第3组	3/67/51	19/3/67	35/19	51/35
第4组	4	20	36	52
⋮	⋮	⋮	⋮	⋮
第15组	15	31	47	63

图 4.3 题 11 中 cache 组和主存块之间的映射

主存的第 0～15 块分别对应 cache 的第 0～15 组，可以放在对应组的任意一行中，在此假定按顺序存放在对应组的第 0 行；主存的第 16～31 块也分别对应 cache 的第 0～15 组，假定放在各组的第 1 行中；同理，主存的第 32～47 块分别放在 cache 的第 0～15 组的第 2 行中；主存的第 48～63 块分别放在 cache 的第 0～15 组的第 3 行中。这样，访问主存的第 0～63 块都没有冲突，每块都是第一次在 cache 中没有找到，然后把这一块调到 cache 对应组的某一行中，这样该块后面的每次访问都能在 cache 中找到。因此，每一块只有第一单元未命中，其余 63 个单元都命中。主存的第 64～67 块分别对应 cache 的第 0～3 组，此时，这 4 组的 4 个行都已经被主存块占满，所以这 4 组的每一组都要选择一个主存块从 cache 中淘汰出来。因为采用 LRU 算法，所以，将最近最少用的第 0～3 块分别从第 0～3 组的第 0 行中替换出来。再把第 64～67 块分别放到 cache 第 0～3 组的第 0 行中，每块也都是第一次在 cache 中未命中，调入后，每次都能在 cache 中命中。

综上所述，第一次循环中，每一块都只有第一单元未命中，其余都命中。

以后的 15 次循环中，因为 cache 第 4～15 组的 48 行中的主存块一直没有被替换过，所以只有 68－48=20 个行中对应主存块的第一个单元未命中，其余都命中。

总访问次数为 4345×16=69520，其中，未命中次数为 68+15×20=368。

命中率 p 为(69520－368)/69520=99.47%。平均访存时间约为 $t_a=t_c+(1-p)\times t_m$=20ns+200×(1－0.9947)ns=20ns+1.06ns=21.06ns。

12. 假定某计算机的 cache 采用直接映射方式，和主存交换的数据块大小为 1 个字，按字编址，一共能存放 16 个字的数据。CPU 开始执行某程序时，cache 为空，在该程序执行过程中，CPU 依次访问以下地址序列：2,3,11,16,21,13,64,48,19,11,3,22,4,27,6 和 11。请回答下列问题：

(1) 每次访问在 cache 中命中还是缺失？试计算访问上述地址序列的 cache 命中率。

(2) 若 cache 数据区容量还是 16 个字，而数据块大小改为 4 个字，则上述地址序列的命中情况又如何？说明块大小和命中率的关系。

【分析解答】

(1) cache 采用直接映射,每行存放一个字,因此共 16 行;每个主存块对应 1 个字,所以主存块号=字号。得到映射公式为 cache 行号=字号 mod 16。程序开始执行时 cache 为空,所以每个单元第一次访问总是缺失(Miss)。CPU 访问给定地址序列的过程如下。(每个数字对“x-y”的含义为“x 是访问的主存地址,y 是对应的 cache 行号”;Hit 表示命中,Miss 表示缺失,Miss/Replace 表示缺失并替换)

2-2:Miss; 3-3:Miss; 11-11:Miss; 16-0:Miss; 21-5:Miss; 13-13:Miss; 64-0:Miss/Replace; 48-0:Miss/Replace; 19-3:Miss/Replace; 11-11:Hit; 3-3:Miss/Replace; 22-6:Miss; 4-4:Miss; 27-11:Miss/Replace; 6-6:Miss/Replace; 11-11:Miss/Replace。只有一次命中。命中率为 1/16=6.25%。

(2) 数据块大小改为 4 个字,cache 能存放 16 个字的数据,故 cache 共 4 行;每个主存块有 4 个字,即主存块号=[字号/4],映射公式为 cache 行号=主存块号 mod 4。CPU 访问给定地址序列的过程如下。(每个数字对“x-y-z”的含义为“x 是访问的主存地址,y 是对应的主存块号,z 是对应的 cache 行号”;Hit 表示命中,Miss 表示缺失,Miss/Replace 表示缺失并替换)

2-0-0:Miss; 3-0-0:Hit; 11-2-2:Miss; 16-4-0:Miss/Replace; 21-5-1:Miss; 13-3-3:Miss; 64-16-0:Miss/Replace; 48-12-0:Miss/Replace; 19-4-0:Miss/Replace; 11-2-2:Hit; 3-0-0:Miss/Replace; 22-5-1:Hit; 4-1-1:Miss/Replace; 27-6-2:Miss/Replace; 6-1-1:Hit; 11-2-2:Miss/Replace。共命中 4 次,命中率为 4/16=25%。数据块变大后,命中率提高了,其原因在于块变大后空间局部性优势得到更大发挥。

13. 假定数组元素在主存按从左到右的下标顺序存放。试改变下列函数中循环的顺序,使得其数组元素的访问与排列顺序一致,并说明为什么修改后的程序比原来的程序执行时间短。

```
int sum_array (int a[N][N][N])
{
    int i, j, k, sum=0;
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            for (k=0; k<N; k++)
                sum+=a[k][i][j];
    return sum;
}
```

【分析解答】

数组元素的访问顺序和排列顺序一致的程序如下:

```
int sum_array (int a[N][N][N])
{
    int i, j, k, sum=0;
    for (k=0; k<N; k++)
        for (i=0; i<N; i++)
            for (j=0; j<N; j++)
```

```
            sum+=a[k][i][j];
    return sum;
}
```

当被访问的数组元素不在 cache 中时，则将该数组元素所在的一个主存块全部装入 cache，因为访问顺序和排列顺序一致，所以，随后访问的若干个数组元素都和该数组元素在同一个主存块中，因而也都能在 cache 中命中。因此，修改后的程序，其数组访问的空间局部性比原程序更好，命中率更高，使得执行时间更短。

14. 某计算机的主存地址空间大小为 256 MB，按字节编址。指令 cache 和数据 cache 分离，均有 8 个 cache 行，主存与 cache 交换的块大小为 64 B，数据 cache 采用直接映射方式。现有两个功能相同的程序 A 和 B，其伪代码如图 4.4 所示。

程序 A:	程序 B:

int a[256][256];
…
int sum_array1 ( )
{
    int i, j, sum=0;
    for (i=0; i<256; i++)
        for (j=0; j<256; j++)
            sum+=a[i] [j];
    return sum;
}
	int a[256][256];
	…
	int sum_array2 ( )
	{
	    int i, j, sum=0;
	    for (j=0; j<256; j++)
	        for (i=0; i<256; i++)
	            sum+=a[i] [j];
	    return sum;
	}

图 4.4　题 14 的伪代码程序

假定 int 类型数据用 32 位补码表示，程序编译时 i，j，sum 均分配在寄存器中，数组 a 按行优先方式存放，其首地址为 320(十进制数)。请回答下列问题，要求说明理由或给出计算过程。

(1) 若不考虑用于 cache 一致性维护和替换算法的控制位，则数据 cache 的总容量为多少?

(2) 数组元素 $a[0][31]$ 和 $a[1][1]$ 各自所在的主存块对应的 cache 行号分别是多少(cache 行号从 0 开始)?

(3) 程序 A 和 B 的数据访问命中率各是多少? 哪个程序的执行时间更短?

【分析解答】

(1) cache 中的每一行信息除了用于存放主存块的数据区外，还有有效位、标记信息，以及用于 cache 一致性维护的修改位(Dirty Bit)和用于替换算法的使用位(如 LRU 位)等控制位。因为主存地址空间大小为 256MB，因而主存地址为 28 位，其中 6 位为块内地址，3 位为行号(行索引)，标志信息有 28－6－3＝19 位。因此，在不考虑用于 cache 一致性维护和替换算法的控制位的情况下，数据 cache 的总容量为 8×(19＋1＋64×8)＝4256 位＝532 字节。

(2) 解法一：要得到某个数组元素所在块对应的 cache 行号，最简单的做法就是把该数

组元素的地址计算出来，然后根据地址求出主存块号，最后用主存块号除以 8 取余数(即主存块号 mod 8)，所得结果就是对应的 cache 行号。因为每个数组元素为一个 32 位 int 型变量，故占 4 个字节。a[0][31]的地址为 320+4×31=444，[444/64]=6，因此a[0][31]对应的主存块号为 6。因为“6 mod 8=6”，所以对应的 cache 行号为 6。

解法二：也可以将地址转换为 28 位二进制地址，然后取出其中的行索引(行号)字段，得到对应行号。首先将地址 444 转换为二进制表示为 0000 0000 0000 0000 000 110 111100，中间 3 位 110 为行索引，因此，对应的 cache 行号为 6。

解法三：用画图的方式可以清楚地表示 cache 行和主存块之间的映射关系。(略)

同理，数组元素 a[1][1]所在主存块对应的 cache 行号为 [(320+4×(1×256+1))/64] mod 8=5。

(3) 编译时 i,j,sum 均分配在寄存器中，故数据访问命中率仅需要考虑数组 a 的访问情况。

① 程序 A 的数据访问命中率。

解法一：由于程序 A 中数组访问顺序与存放顺序相同，故依次访问的数组元素位于相邻单元；该程序共访问 256×256=64K 次数组元素，占 64K×4B/64B=4K 个主存块；因为首地址正好位于一个主存块的边界，故每次将一个主存块装入 cache 时，总是第一个数组元素缺失，其他都命中，共缺失 4K 次，因此，数据访问的命中率为 (64K－4K) /64K=93.75%。

解法二：因为每个主存块的命中情况都一样，因此，也可以按每个主存块的命中率计算。主存块大小为 64B，包含 16 个数组元素，因此，共访存 16 次，其中第一次不命中，所以命中率为 15/16=93.75%。

② 程序 B 的数据访问命中率。

由于程序 B 中的数组访问顺序与存放顺序不同，依次访问的数组元素分布在相隔 256×4=1024 的单元处，因此，依次访问的前后数组元素都不在同一个主存块中；因为数据 cache 只有 8 行，而每次内循环要调入 256×4B/64B=16 个主存块，因此，以前被装入 cache 的主存块，当需要再次访问其中的数组元素时，已经被替换出 cache，因而不能命中。由此可知，所有访问都不命中，命中率为 0。

因为程序 A 的命中率高，因此，程序 A 的执行速度比程序 B 快。

15. 图 4.5 给出了 3 个函数，分析比较它们的空间局部性，并指出哪个最好，哪个最差？

【分析解答】

对于函数 clear1，其数组访问顺序与在内存的存放顺序完全一致，因此，空间局部性最好。

对于函数 clear2，其数组访问顺序在每个数组元素内跳跃式访问，相邻两次访问的单元最大相差 3 个 int 型变量(假定 sizeof(int)=4，则相当于 12B)，因此空间局部性比 clear1 差。若主存块大小比 12B 小，则大大影响命中率。

对于函数 clear3，其数组访问顺序与在内存的存放顺序不一致，相邻两次访问的单元都相差 6 个 int 型变量(假定 sizeof(int)=4，则相当于 24B)因此，空间局部性比 clear2 还差。若主存块大小比 24B 小，则大大影响命中率。

```c
#define N 1000
typedef  struct {
          int vel[3];
          int acc[3];
    } point;
point p[N];
void clear1(point * p, int n)
{
    int i, j;
    for (i=0; i<n; i++) {
        for (j=0; j<3; j++)
            p[i].vel[j]=0;
        for (j=0; j<3; j++)
            p[i].acc[j]=0;
    }
}
```

```c
#define N 1000
typedef  struct {
          int vel[3];
          int acc[3];
    } point;
point p[N];
void clear2(point * p, int n)
{
    int i, j;
    for (i=0; i<n; i++) {
        for (j=0; j<3; j++) {
            p[i].vel[j]=0;
            p[i].acc[j]=0;
        }
    }
}
```

```c
#define N 1000
typedef  struct {
          int vel[3];
          int acc[3];
    } point;
point p[N];
void clear3(point * p, int n)
{
    int i, j;
    for (j=0; j<3; j++) {
        for (i=0; i<n; i++)
            p[i].vel[j]=0;
        for (i=0; i<n; i++)
            p[i].acc[j]=0;
    }
}
```

图 4.5　题 15 的伪代码程序

16. 以下是计算两个向量点积的程序段：

```c
float dotproduct (float x[8], float y[8])
{
    float sum=0.0;
    int i;
    for (i=0; i<8; i++)
        sum+=x[i] *  y[i];
    return sum;
}
```

请回答下列问题。

(1) 访问数组 x 和 y 时的时间局部性和空间局部性各如何？你能否推断出命中率的高低？

(2) 假定数据 cache 采用直接映射方式，数据区容量为 32 字节，每个主存块大小为 16 字节；编译器将变量 sum 和 i 分配在寄存器中，数组 x 存放在 0000 0040H 开始的 32 字节的连续存储区中，数组 y 则紧跟在 x 后进行存放。该程序数据访问的命中率是多少？要求说明每次访问时 cache 的命中情况。

(3) 将上述(2)中的数据 cache 改用 2-路组相联映射方式，块大小改为 8 字节，其他条件不变，则该程序数据访问的命中率是多少？

(4) 在上述(2)中条件不变的情况下，将数组 x 定义为 float[12]，则数据访问的命中率是多少？

【分析解答】

(1) 数组 x 和 y 都按存放顺序访问，空间局部性都较好，但每个数组元素都只被访问一次，故没有时间局部性。命中率的高低与块大小、映射方式等都有关，所以，无法推断命中率的高低。

(2) cache 共有 32B/16B＝2 行；4 个数组元素占一个主存块；数组 x 的 8 个元素(共 32B)分别存放在主存 40H 开始的 32 个单元中，共占有 2 个主存块，其中 $x[0]$～$x[3]$在第 4 块，$x[4]$～$x[7]$在第 5 块中；数组 y 的 8 个元素分别在主存第 6 块和第 7 块中。所以，$x[0]$～$x[3]$和 $y[0]$～$y[3]$都映射到 cache 第 0 行；$x[4]$～$x[7]$和 $y[4]$～$y[7]$都映射到 cache 第 1 行。因为 $x[i]$和 $y[i]$($0\leqslant i\leqslant 7$)总是映射到同一个 cache 行，相互淘汰对方，故每次都不命中，命中率为 0。

(3) 若 cache 改用 2-路组相联，块大小改为 8B，则 cache 共有 4 行，每组两行，共两组。两个数组元素占一个主存块。数组 x 占 4 个主存块，数组元素 $x[0]$～$x[1]$、$x[2]$～$x[3]$、$x[4]$～$x[5]$、$x[6]$～$x[7]$分别在第 8～11 块中；数组 y 占 4 个主存块，数组元素 $y[0]$～$y[1]$、$y[2]$～$y[3]$、$y[4]$～$y[5]$、$y[6]$～$y[7]$分别在第 12～15 块中；因为每组有两行，所以 $x[i]$和 $y[i]$($0\leqslant i\leqslant 7$)虽然映射到同一个 cache 组，但可以存放到同一组的不同 cache 行内，因此，不会发生冲突。每调入一个主存块，装入的 2 个数组元素中，第 2 个数组元素总是命中，故命中率为 50%。

(4) 将数组 x 定义为 12 个元素后，则 x 共有 48B，使得 y 从主存第 7 块开始存放，即 $x[0]$～$x[3]$在第 4 块，$x[4]$～$x[7]$在第 5 块，$x[8]$～$x[11]$在第 6 块中，$y[0]$～$y[3]$在第 7 块，$y[4]$～$y[7]$在第 8 块。因而，$x[i]$和 $y[i]$($0\leqslant i\leqslant 7$)就不会映射到同一个 cache 行中。每调入一个主存块，装入 4 个数组元素，第一个元素不命中，后面 3 个总命中，故命中率为 75%。

17. 以下是对矩阵进行转置的程序：

```
typedef int array[4][4];
void transpose(array dst, array src)
{
    int i, j;
    for (i=0; i<4; i++)
        for (j=0; j<4; j++)
            dst[j][i]=src[i][j];
}
```

假设该程序运行的计算机中 sizeof(int)＝4，且只有一级 cache，其中 L1 data cache 的数据区大小为 32B，采用直接映射、回写方式，块大小为 16B，初始为空。数组 *dst* 从地址 0000 C000H 开始存放，数组 *src* 从地址 0000 C040H 开始存放。仿照 *col*＝0，*row*＝0 栏目中的形式填写表 4.1，说明数组元素 *src*[*row*][*col*]和 *dst*[*row*][*col*]各自映射到 cache 哪一行，其访问是命中(Hit)还是缺失(Miss)。若 L1 data cache 的数据区容量改为 128B 时，重新填写表中内容。

表 4.1 题 17 的 *src* 数组和 *dst* 数组

	src 数组				*dst* 数组			
	col＝0	*col*＝1	*col*＝2	*col*＝3	*col*＝0	*col*＝1	*col*＝2	*col*＝3
row＝0	0/Miss							
row＝1								
row＝2								
row＝3								

【分析解答】

从程序来看，数组访问过程如下：

src[0] [0]、*dst*[0] [0]、*src*[0] [1]、*dst*[1] [0]、*src*[0] [2]、*dst*[2] [0]、*src*[0] [3]、*dst*[3] [0]
src[1] [0]、*dst*[0] [1]、*src*[1] [1]、*dst*[1] [1]、*src*[1] [2]、*dst*[2] [1]、*src*[1] [3]、*dst*[3] [1]
src[2] [0]、*dst*[0] [2]、*src*[2] [1]、*dst*[1] [2]、*src*[2] [2]、*dst*[2] [2]、*src*[2] [3]、*dst*[3] [2]
src[3] [0]、*dst*[0] [3]、*src*[3] [1]、*dst*[1] [3]、*src*[3] [2]、*dst*[2] [3]、*src*[3] [3]、*dst*[3] [3]

因为块大小为16B，每个数组元素有4个字节，所以4个数组元素占一个主存块，因此每次总是调入4个数组元素到cache的一行中。

当数据区容量为32B时，L1 data cache中共有2行。因为地址0000 C000H和0000 C040H的最低5位都是0，所以数组元素*dst*[0][*i*]、*dst*[2][*i*]、*src*[0][*i*]、*src*[2][*i*] (*i*=0～3)都映射到cache第0行，数组元素*dst*[1][*i*]、*dst*[3][*i*]、*src*[1][*i*]、*src*[3][*i*] (*i*=0～3)都映射到cache第1行。因此，从上述访问过程来看，*src*[0][0]所在的主存块(即存放*src*[0][*i*] (*i*=0～3)中4个数组元素的主存块)刚调入cache，*dst*[0][0]所在主存块又把它替换掉了。

当数据区容量为128B时，L1 data cache中共有8行。数组元素*dst*[0][*i*]、*dst*[1][*i*]、*dst*[2][*i*]、*dst*[3][*i*]、*src*[0][*i*]、*src*[1][*i*]、*src*[2][*i*]、*src*[3][*i*]　(*i*=0～3)分别映射到cache第0、1、2、3、4、5、6、7行。因此，不会发生数组元素的替换。每次总是第一个数组元素不命中，后面3个数组元素都命中。

表4.2给出了cache数据容量分别为32B和128B时数组*src*和*dst*的每个元素的命中情况。

表4.2　题17的*src*数组和*dst*数组的命中情况

	*src*数组				*dst*数组			
32B	*col*=0	*col*=1	*col*=2	*col*=3	*col*=0	*col*=1	*col*=2	*col*=3
row=0	0/Miss	0/Miss	0/Hit	0/Miss	0/Miss	0/Miss	0/Miss	0/Miss
row=1	1/Miss	1/Hit	1/Miss	1/Hit	1/Miss	1/Miss	1/Miss	1/Miss
row=2	0/Miss	0/Miss	0/Hit	0/Miss	0/Miss	0/Miss	0/Miss	0/Miss
row=3	1/Miss	1/Hit	1/Miss	1/Hit	1/Miss	1/Miss	1/Miss	1/Miss
	*src*数组				*dst*数组			
128B	*col*=0	*col*=1	*col*=2	*col*=3	*col*=0	*col*=1	*col*=2	*col*=3
row=0	4/Miss	4/Hit	4/Hit	4/Hit	0/Miss	0/Hit	0/Hit	0/Hit
row=1	5/Miss	5/Hit	5/Hit	5/Hit	1/Miss	1/Hit	1/Hit	1/Hit
row=2	6/Miss	6/Hit	6/Hit	6/Hit	2/Miss	2/Hit	2/Hit	2/Hit
row=3	7/Miss	7/Hit	7/Hit	7/Hit	3/Miss	3/Hit	3/Hit	3/Hit

18. 通过对方格中每个点设置相应的CMYK值就可以将方格涂上相应的颜色。图4.6所示的3个程序都可实现对一个8×8的方格涂上黄颜色的功能。

假设cache的数据区大小为512B，采用直接映射，块大小为32B，存储器按字节编址。

<pre>struct pt_color {
        int c;
        int m;
        int y;
        int k;
}
struct pt_color square[8][8];
int i, j;
for (i=0; i<8; i++) {
    for (j=0; j<8; j++) {
        square[i][j].c=0;
        square[i][j].m=0;
        square[i][j].y=1;
        square[i][j].k=0;
    }
}</pre>
	<pre>struct pt_color {
	        int c;
	        int m;
	        int y;
	        int k;
	}
	struct pt_color quare[8][8];
	int i, j;
	for (i=0; i<8; i++) {
	    for (j=0; j<8; j++) {
	        square [j] [i].c=0;
	        square [j] [i].m=0;
	        square [j] [i].y=1;
	        square [j] [i].k=0;
	    }
	}</pre>
		<pre>struct pt_color {
		        int c;
		        int m;
		        int y;
		        int k;
		}
		struct pt_color square[8][8];
		int i, j;
		for (i=0; i<8; i++)
		    for (j=0; j<8; j++)
		        square[i][j].y=1;
		for (i=0; i< 8 ; i++)
		    for (j=0; j< 8 ; j++) {
		        square[i][j].c=0;
		        square[i][j].m=0;
		        square[i][j].k=0;
		    }</pre>

(a) 程序段A	(b) 程序段B	(c) 程序段C

图 4.6　题 18 的伪代码程序

编译时变量 i 和 j 分配在寄存器中，sizeof(int)＝4，数组 *square* 按行优先方式存放在 0000 0C80H 开始的连续区域中，主存地址为 32 位。要求：

(1) 对 3 个程序段 A、B、C 中数组访问的时间局部性和空间局部性进行分析比较。

(2) 画出主存中的数组元素和 cache 行的对应关系图。

(3) 计算 3 个程序段 A、B、C 中数组访问的写操作次数、写不命中次数和写缺失率。

【分析解答】

(1) 程序段 A、B 和 C 中，每个数组元素都是只被访问一次，所以都没有时间局部性；程序段 A 访问顺序和存放顺序一致，所以，空间局部性好；程序段 B 访问顺序和存放顺序不一致，所以，空间局部性不好；程序段 C 的访问顺序和存放顺序部分一致，所以空间局部性的优劣介于程序 A 和 B 之间。

(2) cache 行数为 512B/32B＝16；数组首址为 0C80H＝0000 1100 1000 0000B，正好是主存第 1100100B(100)块的起始地址，所以数组从主存第 100 块开始存放。一个数组元素占用的空间大小为 4×4B＝16B，每 2 个数组元素占用一个主存块，8×8 的数组共占用 32 个主存块，正好是 cache 数据区大小的 2 倍。

因为 100 mod 16＝4，所以主存第 100 块映射到的 cache 行号为 4。

主存中数组元素与 cache 行的映射关系如图 4.7 所示。

(3) 对于程序段 A：每两个数组元素(共涉及 8 次写操作)装入一个 cache 行中，总是第一次访问时未命中，后面 7 次都命中，所以，总的写操作次数为 64×4＝256 次，写不命中次数为 256×1/8＝32 次，因而写缺失率为 12.5%。对于程序段 B：每两个数组元素(共涉及 8 次写操作)装入一个 cache 行中，但总是只有一个数组元素(涉及 4 次写操作)在被淘汰之前被访问，并且总是第一次不命中，后面 3 次命中。即写不命中次数为 256×1/4＝64 次，因而

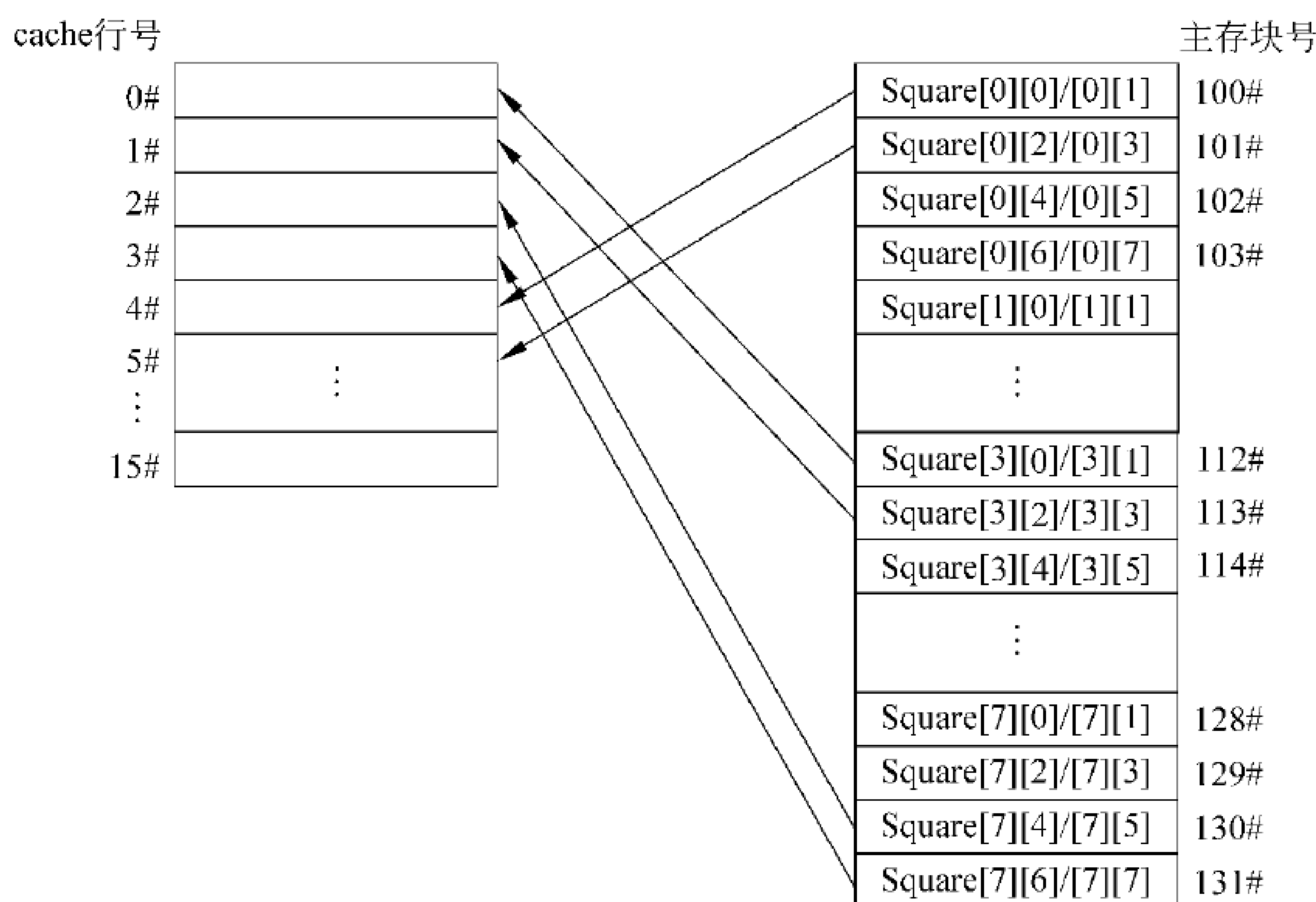


图 4.7　主存中数组元素与 cache 行的映射关系

写缺失率为 25%。对于程序段 C：第一个循环共访问 64 次，每次装入两个数组元素，第一次不命中，第二次命中；第二个循环，共访问 64×3 次，每两个数组元素(共涉及 6 次写操作)装入一个 cache 行中，并且总是第一次不命中，后面 5 次命中。所以总的写不命中次数为 32+(3×64)×1/6=64 次，因而总的写缺失率为 25%。

19. 已知 cache1 采用直接映射方式，共 16 行，块大小为 1 个字，缺失损失为 8 个时钟周期；cache2 也采用直接映射方式，共 4 行，块大小为 4 个字，缺失损失为 11 个时钟周期。假定开始时 cache 为空，采用字编址方式。要求找出一个访问地址序列，使得 cache2 具有更低的缺失率，但总的缺失损失反而比 cache1 大。

【分析解答】

假设 cache1 和 cache2 的缺失次数分别为 x 和 y，根据题意，x 和 y 必须满足以下条件：$11y>8x$ 且 $x>y$，显然，满足该条件的 x 和 y 有许多，例如，$x=4$，$y=3$ 或 $x=5$，$y=4$ 等等。

对于以下的访问地址序列：0，1，4，8，cache1 缺失 4 次，而 cache2 缺失 3 次；

对于以下的访问地址序列：0，2，4，8，12，cache1 缺失 5 次，而 cache2 缺失 4 次；

对于以下的访问地址序列：0，3，4，8，12，16，20，cache1 缺失 7 次，而 cache2 缺失 6 次；

如此等等，可以找出很多。

20. 提高关联度通常会降低缺失率，但并不总是这样。请给出一个地址访问序列，使得采用 LRU 替换算法的 2-路组相联映射 cache 比具有同样大小的直接映射 cache 的缺失率更高。

【分析解答】

2-路组相联 cache 的组数是直接映射 cache 的行数的一半，所以，可以找到一个地址序列 A、B、C，使得 A 映射到某一个 cache 行，B 和 C 同时映射到另一个 cache 行，并且 A、B、C 映射到同一个 cache 组。这样，如果访存的地址序列为 A、B、C、A、B、C、A、B、C、…，则对于直接映射 cache，其命中情况为 Miss/Miss/Miss/Hit/Miss/Miss/Hit/Miss/Miss/…命中率

可达33.3%；对于组相联cache，因为A、B、C映射到同一个组，每组只有2行，采用LRU替换算法，所以，每个地址处的数据刚调出cache就又被访问到，每次都是Miss，命中率为0。例如：假定直接映射cache为4行×1字/行，同样大小的2-路组相联cache为2组×2行/组×1字/行，当访问序列为0、2、4、0、2、4、0、2、4、…(局部块大小为3)时，则出现上述情况。

当访问的局部块比组大时，可能会发生"颠簸(抖动)"现象：刚被替换出去的数据又被访问，导致缺失率为100%！

21. 假定有3个处理器，分别带有以下不同的cache：

cache1：采用直接映射方式，块大小为1个字，指令和数据的缺失率分别为4%和6%；

cache2：采用直接映射方式，块大小为4个字，指令和数据的缺失率分别为2%和4%；

cache3：采用2-路组相联映射方式，块大小为4个字，指令和数据的缺失率分别为2%和3%。

在这些处理器上运行相同的程序，该程序的CPI为2.0，其中有一半是访存指令。若缺失损失为"块大小+6"个时钟周期，处理器1和处理器2的时钟周期都为420ps，带有cache3的处理器3的时钟周期为450ps。请问：哪个处理器因cache缺失而引起的额外开销最大？哪个处理器执行速度最快？

【分析解答】

假设所运行的程序共执行N条指令，每条访存指令仅读写一次内存数据，则在该程序执行过程中各处理器因cache缺失而引起的额外开销和执行时间计算如下。

对于处理器1：额外开销为$N\times(4\%+6\%\times50\%)\times(1+6)=0.49N$个时钟周期，执行程序所需时间为$(N\times2.0+0.49N)\times420\text{ps}=1045.8N(\text{ps})$；

对于处理器2：额外开销为$N\times(2\%+4\%\times50\%)\times(4+6)=0.40N$个时钟周期，执行程序所需时间为$(N\times2.0+0.40N)\times420\text{ps}=1008N(\text{ps})$；

对于处理器3：额外开销为$N\times(2\%+3\%\times50\%)\times(4+6)=0.35N$个时钟周期，执行程序所需时间为$(N\times2.0+0.35N)\times450\text{ps}=1057.5N(\text{ps})$。

由此可见，处理器1 cache缺失引起的额外开销最大，处理器2的执行速度最快。

22. 假定某处理器带有一个数据区容量为256B的cache，其块大小为32B。以下C语言程序段运行在该处理器上，sizeof(int)=4，编译器将变量i,j,c,s都分配在通用寄存器中，因此，只要考虑数组元素的访存情况。若cache采用直接映射方式，则当$s=64$和$s=63$时，缺失率分别为多少？若cache采用2-路组相联映射方式，则当$s=64$和$s=63$时，缺失率又分别为多少？

```
int  i, j, c, s, a[128];
  …
for (i=0; i<10000; i++)
    for (j=0; j<128; j=j+s)
        c=a[j];
```

【分析解答】

因为块大小为32B，所以每个主存块包含8个数组元素；cache共有256B/32B=8行。以下仅考虑数组访问情况。

(1) 直接映射，$s=64$：访存顺序为 $a[0]$、$a[64]$；$a[0]$、$a[64]$、…，共循环 10000 次。这两个元素被映射到同一个 cache 行中，每次都会发生冲突，因此缺失率为 100%。

(2) 直接映射，$s=63$：访存顺序为 $a[0]$、$a[63]$、$a[126]$；$a[0]$、$a[63]$、$a[126]$、…，循环 10000 次。这 3 个元素中后面两个元素因为都映射到同一个 cache 行中，因此每次都会发生冲突，而 $a[0]$不会发生冲突，故缺失率约为 67%。

(3) 2-路组相联，$s=64$：访存顺序为 $a[0]$、$a[64]$；$a[0]$、$a[64]$、…，共循环 10000 次。这两个元素虽然映射到同一个组中，但可以放在该组不同的 cache 行中，因此，仅开始两次缺失，缺失率近似为 0。

(4) 2-路组相联，$s=63$：访存顺序为 $a[0]$、$a[63]$、$a[126]$；$a[0]$、$a[63]$、$a[126]$、…，共循环 10000 次。这 3 个元素中后面两个元素虽映射到同一个组中，但可放在不同 cache 行中，而 $a[0]$不会发生冲突，因此仅开始 3 次缺失，缺失率近似为 0。

23. 假定一个分页虚拟存储系统的虚拟地址为 40 位，物理地址为 36 位，页大小为 16KB，按字节编址。若页表中有有效位、存储保护位、修改位、使用位共占 4 位，磁盘地址不在页表中，则该存储系统中每个进程的页表大小为多少？如果按计算出来的实际大小构建页表，则会出现什么问题？

【分析解答】

因为每页大小有 16KB，所以虚拟页数为 $2^{40}\text{B}/16\text{KB}=2^{(40-14)}=2^{26}$ 页。物理页面和虚拟页面大小相等，所以物理页号(实页号)的位数为 $36-14=22$ 位。每个页表项包括有效位、保护位、修改位、使用位、物理页号等，所以其位数至少为 $4+22=26$。为了简化对页表项的访问，每个页表项取 32 位。因此，每个进程的页表大小为 $2^{26}\times 32\text{b}=256\text{MB}$。如果按实际计算出的页表大小构建页表，则构建出的页表会因为过大而导致页表无法一次装入内存。

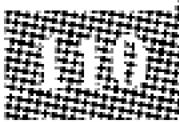
24. 假定一个分页虚拟存储系统按字节编址，逻辑地址有 36 位，页大小为 16KB，物理地址位数为 32 位，页表中有效位和修改位各占 1 位、使用位和存取方式位各占 2 位，而且所有虚拟页都在使用中。请问：每个进程的页表大小至少为多少？如果所使用的快表(TLB)中总的表项数为 256 项，采用 2-路组相联 cache 实现，则快表的大小至少为多少？

【分析解答】

因为页大小为 16KB，所以页内地址位数为 14 位。逻辑地址为 36 位，所以虚页号位数为 $36-14=22$，虚拟页数为 2^{22} 个。因此每个进程的页表项数为 2^{22} 个。物理地址为 32 位，所以物理页号位数为 $32-14=18$。因此每个页表项的位数为 $1+1+2+2+18=24$ 位。所以每个进程的页表大小至少为 $24\text{b}\times 2^{22}=24\times 4\text{Mb}=12\text{MB}$。

TLB 中总的表项数为 256 项，采用 2-路组相联，所以共有 128 组。22 位虚拟页号中低 7 位用来表示组号，高 15 位用来作为标记，和每组中的标记进行比较，以判断是否 TLB 命中。所以，TLB 中每个页表项的位数比主存中页表项的位数多了 15 位的标记，即 TLB 中每个页表项的位数至少为 $24+15=39$ 位，故整个快表的大小至少为 $256\times 39=9984$ 位 $=1248$ 字节。

25. 假定一个计算机系统中有一个 TLB 和一个 L1 data cache。该系统按字节编址，虚拟地址 16 位，物理地址 12 位；页大小为 128B，TLB 为 4-路组相联，共有 16 个页表项；L1 data cache 采用直接映射方式，块大小为 4B，共 16 行。在系统运行到某一时刻时，TLB、页



表和 L1 data cache 中的部分内容(用十六进制表示)如图 4.8 所示。

组号	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位	标记	页框号	有效位
0	03	—	0	09	1D	1	00	—	0	07	10	1
1	13	2D	1	02	—	0	04	—	0	0A	—	0
2	02	—	0	08	—	0	06	—	0	03	—	0
3	07	—	0	63	12	1	0A	34	1	72	—	0

(a) TLB(4-路组相联)：4组、16个页表项

虚页号	页框号	有效位
000	08	1
001	03	1
002	14	1
003	02	1
004	—	0
005	16	1
006	—	0
007	07	1
008	13	1
009	17	1
00A	09	1
00B	—	0
00C	19	1
00D	—	0
00E	11	1
00F	0D	1

(b) 部分页表：(开始16项)

行索引	标记	有效位	字节3	字节2	字节1	字节0
0	19	1	12	56	C9	AC
1	—	0	—	—	—	—
2	1B	1	03	45	12	CD
3	—	0	—	—	—	—
4	32	1	23	34	C2	2A
5	0D	1	46	67	23	3D
6	—	0	—	—	—	—
7	10	1	12	54	65	DC
8	24	1	23	62	12	3A
9	—	0	—	—	—	—
A	2D	1	43	62	23	C3
B	—	0	—	—	—	—
C	12	1	76	83	21	35
D	16	1	A3	F4	23	11
E	33	1	2D	4A	45	55
F	—	0	—	—	—	—

(c) L1 data cache：直接映射，共16行，块大小为4B

图 4.8 题 25 的 TLB、页表和 cache 中的部分内容

请回答下列问题：

(1) 虚拟地址中哪几位表示虚拟页号？哪几位表示页内偏移量？虚拟页号中哪几位表示 TLB 标记？哪几位表示 TLB 索引？

(2) 物理地址中哪几位表示物理页号？哪几位表示页内偏移量？在访问 cache 时，物理地址如何划分成标记字段、行索引字段和块内地址字段？

(3) CPU 从地址 067AH 中取出的值为多少？要求对 CPU 读取地址 067AH 中内容的过程进行说明。

【分析解答】

(1) 16 位虚拟地址中低 7 位为页内偏移量，高 9 位为虚页号；虚页号中高 7 位为 TLB 标记，低 2 位为 TLB 组索引。

(2) 12 位物理地址中低 7 位为页内偏移量，高 5 位为物理页号；在访问 cache 时，12 位物理地址中，低 2 位为块内地址，中间 4 位为 cache 行索引，高 6 位为标记。

(3) 地址 067AH＝0000 0110 0111 1010B，所以，虚页号为 000001100B，映射到 TLB 的第 0 组，将 0000011B＝03H 与 TLB 第 0 组的 4 个标记比较，虽然和其中一个相等，但对应

的有效位为0，其余都不等，所以TLB缺失，需要访问主存中的慢表。直接查看000001100B=00CH处的页表项，有效位为1，取出物理页号19H=11001B，和页内偏移1111010B拼接成物理地址110011111010B。由主存物理地址的划分知，标记为110011B，cache行索引为1110B。根据行索引"1110"直接找到cache第14行(第E行)，其有效位为1，且标记为33H=110011B，正好等于物理地址高6位的标记，故命中。因此，根据物理地址最低两位"10"，取出字节2中的内容4AH=01001010B。

第5章 指令系统

5.1 教学目标和内容安排

主要教学目标：

使学生了解高级语言与汇编语言之间、汇编语言与机器语言之间的关系；掌握指令系统设计中有关指令格式、操作数类型、寻址方式、操作类型等内容；了解高级语言源程序如何转换为机器级代码的过程；并深刻理解CISC和RISC之间的差别。

基本学习要求：

(1) 理解引入高级语言、汇编语言和机器语言的目的。

(2) 了解"存储程序"工作方式的内涵。

(3) 了解指令的基本格式及其设计原则。

(4) 理解定长操作码指令的特点。

(5) 理解扩展操作码指令格式的设计方法。

(6) 理解指令寻址和有效地址的概念。

(7) 掌握常见寻址方式(立即、直接、间接、寄存器、寄存器间接、变址、相对、基址和堆栈寻址方式)。

(8) 理解指令中地址码的位数与主存地址空间大小、最小寻址单位之间的关系。

(9) 理解数据寻址和指令寻址的差别。

(10) 理解累加器型指令的特点。

(11) 理解堆栈型指令的特点。

(12) 理解装入/存储型指令的特点。

(13) 理解通用寄存器型指令的特点。

(14) 理解各种类型指令的功能和操作过程。

(15) 理解分支指令、跳转指令、调用指令和返回指令的特点和相互间的区别。

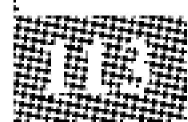
(16) 理解CISC和RISC的区别和各自的特点。

(17) 了解汇编语言和机器语言(指令代码)之间的对应关系。

(18) 了解高级语言源程序和机器语言(指令代码)之间的对应关系。

(19) 了解从高级语言源程序到可执行文件的转换过程。

高级语言源程序最终必须转换成机器级指令代码才能在机器上运行，所以，指令系统最



根本的设计需求来自高级语言，因而，要能够很好地理解指令系统设计涉及的各类问题，最好能够站在高级语言程序员的角度去思考。例如，对于操作数类型，因为在高级语言中有各种不同长度的无符号整数、带符号整数等数据类型，所以底层指令系统中也需要提供相应的不同长度和不同类型数据的支持；对于操作类型，因为高级语言中存在各种运算表达式，以及如循环和选择等各类程序结构，所以，指令系统就必须能够提供不同的运算类指令和不同的程序控制类指令；对于数据的寻址方式，因为高级语言中有简单/复合类型变量、静态/动态变量、全局/局部变量等各种属性数据的存在，使得数据所存放的位置具有多样性，例如，部分简单变量被分配在寄存器中存放，复合类型变量分配在存储器中存放，全局静态变量被分配在存储器的静态数据区，动态变量在存储器的堆区存放，局部变量被分配在存储器的栈区存放，等等，因此，指令需要能够存取在不同地方存放的数据，这就涉及寻址方式问题，所以，寻址方式的多样性是由数据存放位置的多样性决定的。

此外，指令系统的设计需求，还有一部分来自操作系统。操作系统直接和硬件打交道，它通过对 CPU 和 I/O 接口中各种控制和状态寄存器，以及内存中专门的存储区，进行直接的访问来实现对硬件资源的管理，因此，指令系统中还必须提供这些用来对硬件资源直接进行控制和管理的指令，专门供操作系统内核程序所用。

因此，在本章内容的教学过程中，应该适当地结合高级程序设计语言、编译器和操作系统的一些相关内容来讲解，尽量使学生能够在知其然的同时，也知其所以然。

在课时有限的情况下，指令系统实例和程序的机器级表示可以作为课外阅读材料，这部分内容基本上是具体的指令系统，包括 MIPS、Pentium、PowerPC 等处理器的指令系统，以及面向多媒体处理的指令集。在学习了前面基本原理的基础上，学生通过阅读与具体的指令系统相关的资料，可以起到巩固所学，加深理解的作用。

为了方便对指令功能进行形式化描述，通常使用寄存器传送级语言 RTL(Register Transfer Language)，本书所用的 RTL 表示约定为：R[a]表示寄存器 a 的内容，M[a] 表示内存单元 a 的内容，PC 的内容直接用 PC 表示(请参照主教材[①]第 230 页的说明)。

5.2 主要内容提要

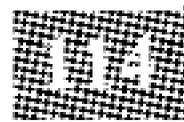
1. 指令格式设计

指令中必须明显或隐含地给出以下信息：操作码、操作数或操作数的地址、结果存放的地址，以及下条指令的地址。通常下条指令地址隐含地由程序计数器 PC 给出。按照指令长度的不同可分为定长指令字格式和变长指令字格式。根据操作码长度的不同，也可分为定长操作码指令格式和变长操作码指令格式。采用定长指令字和定长操作码方式，可以简化指令地址计算以及取指和译码操作，加快指令执行；采用变长指令字和变长操作码方式，则指令更紧凑，使得程序所占空间更少。

2. 操作类型

根据操作类型的不同，可以将指令分成以下几类。数据传送指令：用于数据在寄存器、主存单元、栈顶和 I/O 端口之间进行传送；运算指令：用于各种算术运算和逻辑运算；字符

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)



串处理指令：用于字符串查找、扫描、转换等；I/O 指令：用于 CPU 与外设接口进行数据/状态/命令信息的交换；程序流控制指令：如条件转移、无条件转移、调用、返回等指令；系统控制指令：用于控制机器的启动和停止，进行自愿访管或自陷，以及空操作等。

3. 操作数类型

为了支持高级语言中对不同类型数据的操作，必须提供对不同数据类型进行操作的指令，而且操作数宽度也有多种，以便与高级语言中各种类型数据长度对应，如 8 位、16 位、32 位、64 位等。以 Pentium 处理器 ISA 为例，提供的数据类型有以下几类。

(1) 序数或指针：用 8 位、16 位或 32 位无符号整数表示。

(2) 带符号整数：用 8 位、16 位、32 位或 64 位补码表示。

(3) 实数：用 IEEE 754 浮点数格式表示。

(4) 十进制整数：用 18 位 BCD 码(80 个二进位，其中符号占一个字节)表示。

(5) 字符串：以字节为单位的字符序列，用 ASCII 码表示。

4. 寻址方式

编译器通常将高级语言程序中定义的各类常量和变量所需要的空间分配在不同的寄存器或存储区中，例如，简单变量可能分配在通用寄存器中；静态全局数组变量分配在存储空间的静态数据区；动态变量分配在存储空间的动态数据区(堆区)；局部变量分配在栈区，等等，而且对于数组等复合变量，通常在程序中需要对每个基本元素进行访问和操作。因此，指令系统中必须提供一套行之有效的寻址方式，以方便 CPU 在执行指令时能快速地定位操作数所在位置并取得操作数。通常将操作数所在存储单元的地址称为操作数的有效地址。指令中提供的寻址方式有以下几种。

(1) 立即寻址：指令中直接给出操作数本身。

(2) 直接寻址：指令的地址码给出操作数的有效地址。

(3) 间接寻址：指令的地址码给出操作数有效地址的地址。

(4) 寄存器寻址：指令的地址码给出操作数所在的寄存器编号。

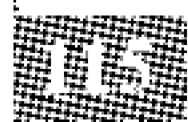
(5) 寄存器间接寻址：指令的地址码给出操作数有效地址所在的寄存器编号。

(6) 堆栈寻址：操作数约定在堆栈中，总是在栈顶取数或存数。

(7) 偏移寻址：用寄存器内容加形式地址得到操作数的有效地址，包括变址寻址、相对寻址和基址寻址 3 种寻址方式。变址寻址方式下，地址码给出一个形式地址，并且隐式或显式地指定一个寄存器作为变址寄存器，变址寄存器的内容(称为变址值)和形式地址相加，得到操作数的有效地址，通常用于循环体中对数组元素的访问；相对寻址方式下，指令中的形式地址给出一个位移量 D，而基准地址由程序计器 PC 提供。通常用于转移指令中转移目标或公共子程序中操作数的寻址；基址寻址方式下，地址码给出的形式地址作为位移量，和基址寄存器的内容相加，得到有效地址。基址寄存器可以在指令中显式指定由某个通用寄存器充当，也可以有一个专门的基址寄存器。

5. 条件码(状态标志)的生成

对应高级语言程序中的选择结构和循环结构，相应的机器代码中需要有条件转移指令，它们根据不同的状态标志来改变程序的执行顺序。通常的状态标志有 CF(进/借位标志)、ZF(零标志)、OF(溢出标志)、SF(符号标志)等。



6. 指令系统风格

按地址码指定风格来分，可以分成累加器型、堆栈型、通用寄存器型和装入/存储型指令系统。累加器型指令系统中，其中一个操作数和运算结果都隐含存放在累加器中，因而指令长度短，但程序的指令条数较多，并需要频繁访问存储器；堆栈型指令系统中，操作数和结果都隐含在堆栈中，因而指令中无须指定地址码，指令长度短，但需频繁访问堆栈，并且对指令序列的顺序要求严格；通用寄存器型指令系统中，操作数明显地指定在通用寄存器中，使用大量通用寄存器，这样既缩短了指令长度，又减少了访问存储器的次数，所以现代计算机大多采用这种指令设计风格；装入/存储型指令系统中，只有装入(Load)指令和存储(Store)指令才能访问内存，而运算类指令的操作数只能在寄存器中，这种类型指令系统本身是通用寄存器型指令系统。

按指令系统的复杂度来分，可分成CISC(复杂指令系统计算机)和RISC(精简指令系统计算机)两类指令系统。CISC指令系统大多采用变长指令字和扩展操作码编码，指令格式多，指令条数多，寻址方式多而复杂，因而指令的译码实现复杂，大多用微程序控制器实现；RISC指令系统大多采用定长指令字和定长操作码，指令格式少，指令系统中仅含有一些常用指令，因而指令条数少，寻址方式少且简单，指令的译码实现简单，可用硬连线路控制器实现。RISC处理器中设置大量的通用寄存器，可大大减少存储器访问次数，并且采用装入/存储型指令设计风格，因而大部分指令的执行步骤一致、规整，指令的执行适合采用流水线方式。

7. MMX和SIMD指令技术

多媒体数据处理中存在大量具有共同特征的操作，可以提供一种同时对多个数据元素进行相同操作的指令，这种指令被称为SIMD(Single Instruction Multi Data)指令。Intel公司于1997年首次提供了一组称为MMX(Multi Media Extensions)技术的指令，这组指令可以对8个64位寄存器中的数据进行处理，每个64位数据可以是8个字节，或4个字，或2个双字，或一个64位的四字。

8. 程序的机器级表示

不管用哪种语言编写的程序最终都必须被转换成用0和1表示的机器代码，也就是指令序列。因为机器代码可读性差，所以引入了与机器语言一一对应的符号化表示语言，称为汇编语言。机器语言和汇编语言都是机器级表示语言。程序的机器级表示主要是编译程序和解释程序的任务。但是，对于指令集体系结构(ISA)的设计者来说，也必须了解高级语言与机器级表示语言之间的对应关系。

通常，高级语言中的赋值语句被转换成由若干运算类指令构成的指令序列；像if语句这样的选择结构程序段被转换为一段包含分支指令(条件转移指令)的指令序列；像for语句这样的循环结构程序段被转换为包含无条件跳转指令和分支指令的被循环执行的指令序列。此外，对于高级语言源程序中的过程调用，也必须在ISA中有相应的支持过程调用的机制，例如，必须提供调用(转子)指令和过程返回指令，同时，因为需要支持嵌套调用和递归调用，所以还必须提供对栈和栈帧的操作指令，等等。

9. 指令系统举例

以MIPS指令集体系结构为例说明。MIPS采用了典型的RISC风格指令系统，提供了32个32位通用寄存器，寄存器编号占5位，各通用寄存器的名称、编号和功能见表5.1。

表 5.1　MIPS 通用寄存器

名　　称	编　　号	功　　能
zero	0	恒为 0
at	1	为汇编程序保留
v0～v1	2、3	过程调用返回值
a0～a3	4～7	过程调用参数
t0～t7	8～15	临时变量，在被调用过程无须保存
s0～s7	16～23	在被调用过程需保存
t8～t9	24、25	临时变量，在被调用过程无须保存
k0～k1	26、27	为 OS 保留
gp	28	全局指针
sp	29	栈指针
fp	30	帧指针
ra	31	过程调用返回地址

通用寄存器的汇编表示可以使用名称(如 $t0)，也可以用编号(如 $0)。此外，MIPS 中还提供了 32 个 32 位浮点寄存器，两个 32 位浮点寄存器构成一个 64 位双精度浮点寄存器。为了实现乘除运算，MIPS 还提供了两个乘商寄存器 Hi 和 Lo，这两个寄存器无须在指令中明显给出，因而，与程序计数器 PC 一样，它们没有寄存器编号。

MIPS 采用三地址指令格式，表 5.2 给出了 MIPS 汇编语言示例列表。

表 5.2　MIPS 汇编语言示例列表

类别	指令名称	汇编举例	含　　义	备　　注
算术运算	add	add $s1,$s2,$s3	$s1=$s2+$s3	三个寄存器操作数
	subtract	sub $s1,$s2,$s3	$s1=$s2−$s3	三个寄存器操作数
存储访问	load word	lw $s1,100($s2)	$s1=M[$s2+100]	从内存取一个字到寄存器
	store word	sw $s1,100($s2)	M[$s2+100]=$s1	从寄存器存一个字到内存
逻辑运算	and	and $s1,$s2,$s3	$s1=$s2 & $s3	三个寄存器操作数，按位与
	or	or $s1,$s2,$s3	$s1=$s2 | $s3	三个寄存器操作数，按位或
	nor	nor $s1,$s2,$s3	$s1=～($s2 | $s3)	三个寄存器操作数，按位或非
	and immediate	andi $s1,$s2,100	$s1=$s2 & 100	寄存器和常数，按位与
	or immediate	ori $s1,$s2,100	$s1=$s2 | 100	寄存器和常数，按位或
	shift left logical	sll $s1,$s2,10	$s1=$s2<<10	按常数对寄存器逻辑左移
	shift right logical	srl $s1,$s2,10	$s1=$s2>>10	按常数对寄存器逻辑右移

续表

类别	指令名称	汇编举例	含义	备注
条件分支	branch on equal	beq \$s1,\$s2,L	if(\$s1==\$s2) go to L	相等则转移
	branch on not equal	bne \$s1,\$s2,L	if(\$s1!=\$s2) go to L	不相等则转移
	set on less than	slt \$s1,\$s2,\$s3	if(\$s2<\$s3) \$s1=1; else \$s1=0	小于则置寄存器为1,否则为0,用于后续指令判0
	set on less than immediate	slti \$s1,\$s2,100	if(\$s2<100) \$s1=1; else \$s1=0	小于常数则置寄存器为1,否则为0,用于后续指令判0
无条件跳转	jump	j L	go to L	直接跳转至目标地址
	jump register	jr \$ra	go to \$ra	过程返回
	jump and link	jal L	\$ra=PC+4; go to L	过程调用

5.3 基本术语解释

指令(Instruction)

是计算机硬件能够识别并直接执行的操作命令。用二进制序列表示,由操作码和地址码两部分组成。

指令系统(Instruction Set)

也称指令集,是计算机中所有指令的集合。

指令集体系结构(Instruction Set Architecture,ISA)

是计算机硬件与系统软件之间的接口,其核心部分是指令集,同时还包含数据类型和数据格式定义、寄存器设计、I/O空间的编址和数据传输方式、中断结构、计算机状态的定义和切换、存储保护等。

指令字长(Instruction Length)

一条指令的二进制代码位数。有定长指令字机器和变长指令字机器。

定长指令(Fixed Length Instruction)

机器中所有指令的位数是相同的,目前定长指令字大多是32位指令字。

变长指令(Variable Length Instruction)

机器的指令有长有短,但每条指令的长度一般都是8的倍数。

操作码(Operate Code)

指令中用于指出操作性质的字段。一般分为定长操作码和扩展操作码。定长操作码指机器中所有指令的操作码字段位数相同。扩展操作码的机器中指令的操作码字段位数不都相同,也称为不定长操作码。

地址码(Address Code)

指令中用于指出操作数地址的字段。一条指令中一般有多个地址码字段。地址码字段的个数与许多因素有关。一个地址码字段可能是一个立即数;可能是一个直接地址;可能是一个间接地址;可能是寄存器编号;可能是I/O端口号;可能是一个形式地址等。

大端序(Big Endian Ordering)

操作数在主存存放时,指令中给出的地址是操作数最高有效字节(MSB)所在的地址。例如,假设一个 32 位数据“12345678H”的地址为 100,则 12H、34H、56H 和 78H 分别存放在第 100、101、102 和 103 号单元中。IBM S/370,Motorola 680x0 等是大端序机器。

小端序(Little Endian Ordering)

操作数在主存存放时,指令中给出的地址是操作数最低有效字节(LSB)所在的地址。例如,假设一个 32 位数据“12345678H”的地址为 100,则 12H、34H、56H 和 78H 分别存放在第 103、102、101 和 100 号单元中。Intel 80x86 等是小端序机器。

字地址(Word Address)

每个主存单元都有一个地址,假定机器中一个字为 32 位,按字节编址,那么字地址就是指具有 4 的倍数的那些地址,如 0、4、8、12、…;对应的还有半字地址(2 的倍数,如 0、2、4、6、…)、双字地址(8 的倍数,如 0、8、16、…)。

边界对齐(Boundary Alignment)

有些机器在操作数存放到主存单元时,要求按照相应的地址边界进行对齐。如假定机器中一个字为 32 位,按字节编址,那么一个 32 位的数据(如一个 float 型的变量或 32 位的 int 型整数变量等)就必须存放在字地址上;一个 16 位的数(如 16 位 short 型的短整数等)就必须放在半字地址上;而 8 位的数据(如 char 型字符)就可以放在任何边界地址上而不需对齐。

累加器(Accumulator)

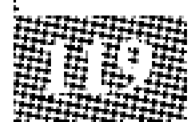
在 CPU 中,累加器(Accumulator,AC 或 ACC) 是一种暂存器,用来储存计算所产生的中间结果。早期机器中没有通用寄存器组,只有一个累加器,这种情况下,如果没有像累加器这样的暂存器,那么在每次计算后就必须要把结果写回到主存,然后可能还要再读回来。这样,就会增加访问主存的次数,降低程序运行的效率。典型的例子就是把一列数字加起来。一开始累加器设定为零,每个数字依序被加到累加器中,当所有数字都被加入后,结果才写回到主存中,而不必每次都写主存。

程序计数器(Program Counter,PC)

又称指令计数器或指令指针 IP,是一个特殊的地址寄存器,专门用来存放下一条要执行指令的地址。因为本身它是个寄存器,所以也称为指令指针寄存器或指令地址寄存器。通常程序是顺序执行的,程序的指令序列在主存中一般也是按连续地址存放的。在开始运行程序之前,总是将第一条指令的地址放入 PC;每取出一条指令并执行后,控制器就使 PC 的内容自动增量(加上当前指令的长度),指明下一条要执行的指令所存放的存储单元地址,从而可以控制指令的顺序执行;在遇到需要改变程序执行顺序的情况时,一般由转移类指令将转移目标地址送往程序计数器,即可实现程序的转移。

指令寄存器(Instruction Register)

指令寄存器(Instruction Register,IR)用来保存当前正在执行的一条指令。当执行一条指令时,先从存储器取出指令,然后送至指令寄存器。指令寄存器中的操作码部分被送到指令译码器 ID(Instruction Decoder)中,经 ID 译码(识别这条指令的功能)后,送出具体的操作控制信号。



程序状态字(Program Status Word,PSW)

表示程序运行状态的一个二进制位序列。一般包含一些反映指令执行结果的标志信息(如进位标志、溢出标志、符号标志等)和设定的一些状态信息(如中断允许/禁止状态、管理程序/用户程序状态等)。

程序状态字寄存器(Program Status Word Register, PSWR)

用来存放程序状态字的寄存器。

标志寄存器(Flags Register)

80x86 体系结构中用来表示程序状态和标志的寄存器。

堆栈(Stack)

堆栈是一块特殊的存储区。采用"先进后出"的方式进行访问。主要用来在程序切换(如过程调用)时保存各种信息。栈底固定不动,栈顶浮动,用一个专门的寄存器(SP)来作为栈顶指针。从堆栈生长的方向来分,有"自顶向下"和"自底向上"两种堆栈。从堆栈的位置来分,有硬堆栈和软堆栈。硬堆栈的堆栈区由寄存器实现,软堆栈的堆栈区由一个主存区域实现。

堆栈指针(Stack Pointer,SP)

是一个特殊的地址寄存器,用来存放堆栈的栈顶指针。如果是硬堆栈,栈顶指针是栈顶寄存器的编号;如果是软堆栈,栈顶指针是栈顶主存单元的地址。

双目运算(Two-operand Operate)

需要两个操作数才能进行的运算。如加、减、乘、除、与、或等算术或逻辑运算都是双目运算。

单目运算(One-operand Operate)

只需要一个操作数就能进行的运算。如取负、取反等算术或逻辑运算都是单目运算。

寻址方式(Addressing Mode)

在程序执行过程中,需要取指令和操作数,确定指令和操作数的存放位置的方式称为寻址方式。确定指令存放位置的过程称为指令寻址,确定操作数存放位置的过程称为数据寻址。

有效地址(Effective Address)

存储器操作数所在存储单元的地址。若不采用虚拟存储机制,则有效地址是主存地址;若采用虚拟存储机制,则有效地址是虚拟地址。

立即寻址(Immediate Addressing)

指令中的地址码直接给出操作数本身。

直接寻址(Direct Addressing)

指令中的地址码给出的是操作数所在的存储单元地址,即有效地址,称为直接地址。

间接寻址(Indirect Addressing)

指令中的地址码给出的是操作数有效地址的地址,称为间接地址。

寄存器寻址(Register Addressing)

指令中的地址码给出的是操作数所在的寄存器的编号。

寄存器间接寻址(Register Indirect Addressing)

指令中的地址码给出的是操作数的有效地址所在的寄存器的编号。

偏移寻址(Displacement Addressing)

指令通过某种方式给出一个形式地址和一个基地址(往往在某个寄存器中),经过相应的计算(基地址加形式地址)得到操作数的有效地址。具体的偏移寻址方式有:变址寻址、相对寻址和基址寻址。

变址寻址(Indexing Addressing)

变址寻址方式下,指令中的地址码给出一个形式地址,并且隐含或明显地指定一个寄存器作为变址寄存器,变址寄存器的内容(变址值)和形式地址相加,得到操作数的有效地址,根据有效地址访问内存,去取操作数或写运算结果。

变址寄存器(Index Register)

是一个特殊的地址寄存器,用来存放变址寻址方式下的变址值,通常是数组元素的下标值或相对数组首地址的偏移量。

相对寻址(Relative Addressing)

相对寻址方式下,指令中的形式地址给出一个位移量 D,而基准地址由程序计数器 PC 提供。即有效地址为 PC 的内容与位移量 D 之和。位移量 D 可正可负,也就是说,需访问的信息可以在当前指令之前 D 个单元处,也可以在当前指令之后 D 个单元处。

基址寻址(Base Addressing)

基址寻址方式下,指令中的地址码给出一个形式地址,作为位移量,并且隐含或明显地指定一个寄存器作为基址寄存器,基址寄存器的内容和形式地址相加,得到操作数的有效地址,根据有效地址访问内存,去取操作数或写运算结果。

基址寄存器(Base Register)

是一个特殊的地址寄存器,用来存放基址寻址方式下的基准地址。通常是一个用户程序在主存的首地址,或一块存储区的首地址。

堆栈寻址(Stack Addressing)

堆栈寻址方式下,操作数被指定在堆栈中。堆栈寻址总是从栈顶取操作数,运算后的结果自动放到栈顶。栈顶的位置由一个专门的堆栈指针 SP 来指示。所以,指令中不需给出操作数地址,是一种零地址指令。与堆栈有关的操作有:入栈(PUSH)、出栈(POP)和运算类操作。

通用寄存器(General Purpose Register,GPR)

一般把用户可访问寄存器称为通用寄存器(GPR)。这些寄存器都有一个编号,在指令中用编号标识寄存器。所以执行指令时,指令中的寄存器编号要送到一个地址译码器进行译码,然后才能选中某个寄存器进行读写。通用寄存器可以用来存放操作数或运算结果,或作为地址指针寄存器、变址寄存器、基址寄存器等。

RR 型指令(Register-Register Type Instruction)

两个操作数都在寄存器中的指令。

RS 型指令(Register-Storage Type Instruction)

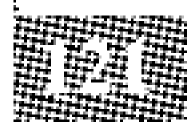
一个操作数在寄存器中,另一个操作数在主存单元中的指令。

SS 型指令(Storage-Storage Type Instruction)

两个操作数都在主存单元中的指令。

数据传送指令(Data Transfer Instruction)

将数据在寄存器和寄存器之间、存储器单元和寄存器之间进行传送的指令。



取数指令(Load)

特指将数据从内存单元取到寄存器的指令。

存数指令(Store)

特指将数据从寄存器保存到内存单元的指令。

相对转移(Relative Jump)

转移目标地址通过PC的值加上一个偏移量形成。所以,转移到的目的地和当前指令的位置有关。

绝对转移(Absolute Jump)

转移目标地址由指令指定的一个绝对地址确定,而与当前指令的位置关系不大。

条件转移(Conditional Jump,Branch)

是一种分支指令,也称为条件分支。根据前面指令或本条指令执行的结果确定是跳转到转移目标地址处执行,还是顺序执行。

无条件转移(Unconditional Jump)

是一种直接跳转指令,执行完本条指令后,无条件地跳转到目标转移地址处执行。

过程(Procedure)

构造过程或子程序是程序员进行模块化程序设计的一种手段,通常程序员把一个大的任务分解成一些子任务,每个子任务用一个过程来实现,这样做,一方面使得程序容易理解,另一方面也使过程可以被多个程序使用,即可重用代码。

过程调用(Procedure Call)

一个过程调用包括将数据(以过程参数和返回值的形式出现)和控制从一个程序传递到另一个程序。此外,在进入过程时,必须为过程的局部变量分配空间,并在退出过程时释放这些空间。

跳转链接指令(Jump and Link Instruction)

用于将控制从调用程序转移到被调用程序的指令。在MIPS指令系统中它称为跳转链接指令jal。在有些机器的指令系统中也被称为过程调用指令(Call Instruction)或转子指令(意思为转移到子程序),如在x86体系结构中的Call指令。在这类调用指令中,需要给出被调用程序的首地址。其主要操作过程为保存返回地址到特定的寄存器或堆栈并跳转到被调用程序。

返回地址(Return Address)

跳转链接指令(调用指令)后面的一条指令的地址。即被调用程序执行完后必须返回的返回点。

返回指令(Return Instruction)

用于从被调用程序控制转回到调用程序的指令。该指令从某个特定的寄存器或当前栈顶取得返回地址,并按返回地址进行跳转。

调用程序(Caller)

在过程调用中,通过过程调用指令调用一个过程或子程序的程序。有时也称为主程序。

被调用程序(Callee)

用过程调用指令(或称为跳转链接指令)所调用的程序。被调用程序最后必须用返回指令返回到调用程序中调用指令后面的那条指令继续执行。

叶过程(Leaf Procedure)

不调用任何过程的过程。

过程帧(Procedure Frame)

过程调用中的参数传递、被调用过程中的局部变量的分配和释放等需要专门的机制来实现。主要是通过栈(Stack)来实现,栈用来传递过程参数、存储返回信息、保存寄存器和过程局部变量等。为单个过程分配的那部分栈区称为过程帧,也称为栈帧(Stack Frame)。

帧指针(Frame Pointer)

在程序运行过程中,可能会有多个过程被嵌套调用,每个未返回的过程都有一个过程帧,当前正在执行的过程帧称为当前帧。它由两个指针来定界,一个是指示当前帧底部的帧指针 fp,另一个是指示当前帧顶部的栈指针 sp。

源程序文件(Source Program File)

用某种高级语言书写的源程序文件。如 C 语言源程序文件 *.c 等。

汇编语言程序文件(Assemble Language Source File)

用某种汇编语言书写的源程序文件。如 UNIX 系统中的汇编语言源程序文件 *.s,或 MS-Windows 中的 *.asm 文件等。

目标程序文件(Object/Target Program File)

编译程序和汇编程序对源程序进行翻译处理所得到的机器语言程序称为目标程序文件,是由机器指令组成的二进制代码。如 UNIX 系统中的 *.o 文件,或 MS-Windows 系统中的 *.obj 文件等。一般而言,目标程序文件中包含目标文件头、文本段(机器代码)、数据段、重定位信息、符号表、调试信息等。因为目标文件是可重定位的,所以也称为可重定位目标文件。

可执行程序文件(Executable Program File)

通过装入程序直接装入存储器执行的文件。一般而言,可执行程序文件除了不包含未确定的调用信息、重定位信息、符号表和调试信息外,它与目标程序文件有相同的格式。可执行程序文件中的模块可以调用其他动态链接库中的函数。

链接程序(Linker/Link Editor)

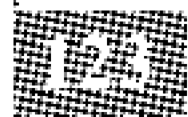
把编译或汇编好的多个目标程序文件和一些库函数目标文件链接生成一个可执行程序文件的工具软件。

装入程序(Loader)

也称为加载器或加载程序。用于将可执行程序文件读入存储器并启动执行的工具软件。一般来说,可执行文件的装入是由操作系统内核来实现的,因此加载程序属于操作系统的一部分。

全局指针(Global Pointer)寄存器

操作系统通过执行装载程序把一个可执行文件装入存储器时,必须按照一定的存储映像使用存储空间。基于 MIPS 处理器的系统通常将存储空间分为 3 个部分,第一部分是从 0x00400000 开始的正文段,用于存放程序代码;第二部分是数据段,分成静态数据段和动态数据段两部分,静态数据段从 0x10000000 开始,其上是动态数据段,随着程序的执行进行动态分配,向上扩展(从低地址向高地址增长);第三部分是程序的堆栈段,处于存储地址空间的最上端,从地址 0x7FFFFFFF 开始向下扩展(从高地址向低地址增长)。MIPS 处理器为



了方便地实现对静态数据区的访问，把地址 0x10008000 放在一个专门的寄存器 $gp 中，将其作为指向静态数据段的全局指针。这样在 Load 和 Store 指令中用 16 位偏移量就可以对静态数据段的前 64KB 的区域进行方便存取。

伪指令(Pseudo Instruction)

汇编语言中使用的但在机器语言中不存在的机器指令。通常在汇编语言源程序中用一条更加简洁自然的伪指令来表示多条机器指令，它们在功能上等价。汇编程序在进行汇编时，将伪指令转换为等价的机器指令序列。

CISC(Complex Instruction Set Computer)

复杂指令集计算机。早期的计算机为了增加功能和更好地支持高级语言而不断地增加新的指令类型，使系统可以实现复杂的操作。这种指令系统功能复杂，寻址方式多，指令长度可变，指令格式多样。因而采用这种指令系统的计算机被称为复杂指令集计算机。

RISC(Reduced Instruction Set Computer)

精简指令集计算机。这种计算机采用简化的指令系统，指令集中只包含程序中常用的指令，运算类指令只能是 R-R 型，提供大量通用寄存器以减少访存次数，采用流水线方式执行指令，控制器用硬连阵列逻辑实现，并采用优化的编译技术。

5.4 常见问题解答

1. 一台计算机中的所有指令都是一样长吗？

答：不一定。有定长指令字机器和不定长指令字机器两种。定长指令字机器中所有指令都一样长，称为规整型指令，目前定长指令字大多是 32 位指令字。不定长指令字机器的指令有长有短，但每条指令的长度一般都是 8 的倍数。所以，一个指令字在存储器中存放时，可能占用多个存储单元；从存储器读出并通过总线传输时，可能分多次进行，也可能一次读多条指令。

2. 每一条指令中都包含操作码吗？

答：是的。每一条指令都必须告诉 CPU 该指令做什么操作，所以必须指定操作码。

3. 每条指令中的地址码个数都一样吗？

答：不一定，有的没有地址码，有的包含一个地址码，有的是两个或三个地址码。地址码个数不一样的主要原因有 3 个：(1)每条指令的操作数个数可能不同。有的指令是双目运算指令，涉及两个源操作数和目操作数，有的是单目运算，只涉及一个源操作数和目操作数，还有的指令只是控制操作，不涉及操作数，如停机、复位、空操作等指令。所以每条指令涉及的操作数个数不同。(2)每个操作数的寻址方式可能不同。而不同的寻址方式给出的地址码个数也不同。(3)地址码的缺省方式可能不同。有的操作数或地址码用的是隐含指定方式，在指令中缺省，不明显给出，如累加器，堆栈等。综上所述，每条指令的地址码个数可能相差很大。

4. 指令中的所有操作数都采用相同的寻址方式吗？

答：不一定。规整型指令一般在一条指令中只包含一种寻址方式，这样，在指令操作码中就隐含了寻址方式，不需要专门有寻址方式字段。但是，对于不规整型指令，一条指令中的若干操作数可能存放在不同地方，因而每个操作数可能有各自的寻址方式。

5. 指令中要明显给出下一条指令的地址吗？

答：不需要。指令在主存中按执行顺序连续存放。大多数情况下指令被顺序执行，只有遇到转移指令（如无条件转移、条件分支、调用和返回等指令）才改变指令执行的顺序。所以，可以用一个专门的计数器，来存放下一条要执行的指令地址，而不需要在指令中专门给出下一条指令的地址。这个计数器称为程序计数器 PC 或指令指针 IP。

当顺序执行时，CPU 直接通过对 PC 加“1”来使 PC 指向下一条顺序执行的指令；当执行到转移指令时，根据指令执行的结果进行相应的地址运算，把运算得到的转移目标地址送到 PC 中，使得执行的下一条指令为转移到的目标指令。

6. 一个操作数在主存中可能占多个单元，怎样在指令中给出操作数的地址呢？

答：现代计算机大多采用字节编址方式，即一个主存单元只能存放一个字节的信息。一个操作数（如 char 型、int 型、float 型、double 型）可能是 8 位、16 位、32 位或 64 位等，因此，可能占用 1 个、2 个、4 个或 8 个主存单元。也就是说，一个操作数可能有多个主存地址对应，在指令中给出哪个地址呢？

有两种不同的地址指定方式：大端方式和小端方式。大端方式下，指令中给出的地址是操作数最高有效字节所在的地址。小端方式下，指令中给出的地址是操作数最低有效字节所在的地址。

7. 地址码位数与主存地址空间大小和编址单位的关系是什么？

答：指令中的地址码如果是主存单元的地址，那么，地址码的位数与主存地址空间大小和编址单位的长度有关。编址单位的长度就是主存单元的宽度，也就是最小的寻址单位。主存可以按字节编址（8 位），也可以按字编址（如 16 位，32 位等）。主存地址空间大小和编址单位确定后，地址码的位数就被确定了。例如，若主存地址空间大小为 4GB，编址单位是字节，则主存单元的地址就是 32 位（$4GB=2^{32}B$）；若按字（假定一个字为 32 位）编址，则主存单元的地址就是 30 位（$4GB=2^{32}B=2^{30}\times 4B$）。

8. 累加器型指令有什么特点？

答：累加器型指令的一个源操作数和目操作数总是在累加器中，是隐含指定的，所以指令中不需要给出累加器的编号。因而，累加器型指令的指令字相对来说较短，但由于每次运算结果都只能放到累加器中，所以可能会增加一些从累加器取数的指令而使程序变长。

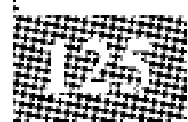
9. 堆栈型指令有什么特点？

答：与堆栈有关的操作有入栈（PUSH）、出栈（POP）和运算类操作。运算类指令分单目运算和双目运算，运算时总是从栈顶取操作数，运算后的结果自动放到栈顶。所以，指令中不需要给出操作数地址。因此，堆栈指令是零地址指令，指令字较短。但因为所有的操作数都只能在栈顶，所以，会增加很多入栈指令而使得程序变长。

堆栈指令的访存次数，取决于采用的是软堆栈还是硬堆栈。如果是软堆栈（堆栈区由主存实现）的话，对于双目运算，需要访存 4 次：取指令、取源操作数 1、取源操作数 2、存结果。如果是硬堆栈（堆栈区由寄存器实现），则只需取指令时访问一次内存。

10. 通用寄存器型指令有什么特点？

答：通用寄存器型指令是相对于累加器型指令和堆栈型指令而言的，指令中的操作数和运算的结果既不是隐含在累加器中，也不是隐含在堆栈中，而是在 CPU 中提供了多个通用寄存器，操作数和结果可以放在这些寄存器中，指令必须明显地指出操作数和结果在哪个



寄存器或哪个主存单元中，要给出寄存器的编号或主存单元地址。目前大多数指令系统采用通用寄存器型指令风格。

11. 装入/存储型指令有什么特点？

答：装入/存储型指令是用在规整型指令系统中的一种通用寄存器型风格指令。为了规整指令格式，使指令具有相同的长度，规定只有装入/存储(Load/Store)指令才能访问内存，而运算指令不能直接访问内存，只能从寄存器取数进行运算，运算的结果也只能送到寄存器。因为，寄存器编号较短，而主存地址位数较长，通过某种方式可以使运算指令和访存指令的长度一致。

这种装入/存储型风格的指令系统最大的特点是指令格式规整，指令长度一致，一般为32位。由于只有Load/Store指令才能访问内存，程序中可能会包含许多装入指令和存储指令，与一般通用寄存器型风格指令相比，其程序长度会更长。

12. 指令寻址方式和数据寻址方式有什么不同？

答：程序被启动时，程序所包含的指令和数据都被装入内存中。在程序中的指令过程中，需要取指令和操作数，确定指令存放位置的过程称为指令寻址，确定操作数存放位置的过程称为数据寻址。指令寻址和数据寻址其复杂度是不一样的。

指令寻址：指令基本上按执行顺序存放在主存中，执行过程中，指令总是从主存单元被取到指令寄存器IR中。顺序执行时，用指令计数器PC加"1"来得到下一条指令的地址；跳转执行时，通过转移指令的寻址方式，计算出目标地址，送到PC中即可。目标转移地址的形成方式主要有3种：立即寻址(直接地址)、相对寻址(相对地址)和间接寻址(间接地址)。

数据寻址：开始时，数据被存放在主存中，但在指令执行过程中，主存数据可能被装入CPU的寄存器中，或者寄存器数据被装入主存的堆栈区中；还有的操作数可能是I/O端口中的内容，或本身就包含在指令中(即立即数)。另外，运行的结果也可能要被送到CPU的寄存器、堆栈、I/O端口或主存单元中，所以，数据的寻址要涉及对寄存器、主存单元、堆栈、I/O端口和立即数的访问。此外，操作数可能是某个一维或多维数组的元素，因此，还要考虑如何提供相应的寻址方式，以方便地在主存中找到数组元素。综上所述，数据的寻址比指令的寻址要复杂得多。

13. 如何指定指令的寻址方式？

答：CPU根据指令约定的寻址方式对地址码的有关信息进行解释，以找到下条要执行的指令，或指令所需要的操作数。有的指令设置专门的寻址方式字段，显式说明采用何种寻址方式，有的指令通过操作码隐含确定寻址方式。

规整型指令一般在一条指令中只包含一种寻址方式，这样，就可在指令操作码中隐含寻址方式，不需要有专门的寻址方式字段。但是，对于不规整型指令，一条指令中的若干操作数可能存放在不同的地方，因而每个操作数可能有各自的寻址方式字段。

14. 指令的操作数可能存放在机器的哪些地方？

答：指令的操作数可能存放在主存单元、寄存器、堆栈、I/O端口中或直接存在于指令本身。(1)主存单元。指令必须以某种方式给出所在单元的地址。又可分为以下几种情况：对单个独立的操作数进行处理；对一个数组中的若干个连续元素或一个数组元素进行处理；对一个表格或表格中的某个元素进行处理，等等。对于这些不同的情况需要提供不同的寻址方式进行操作数的访问。(2)寄存器。指令中只要直接给出寄存器的编号即可。(3)堆栈

区。若有专门的堆栈指令，则指令中不需要给出操作数的地址，数据的地址隐含地由堆栈指针给出。(4)I/O端口。当某个I/O接口中的寄存器内容要和CPU中的寄存器内容交换时，要用I/O指令，在I/O传送指令中，需提供I/O端口号。(5)指令中的立即数。此时，操作数是指令的一部分，直接从指令中的立即数字段取操作数。

15. 有哪些常用的数据寻址方式？

答：数据寻址方式可以归为以下几类。(1)立即寻址。指令中的立即数字段，可以作为操作数，也可以作为直接转移地址。取到ALU运算前，可能要对其进行扩展。(2)直接寻址类。指令中直接给出操作数所在的寄存器编号、I/O端口号或主存单元地址。如直接寻址方式、寄存器寻址方式。(3)间接寻址类。操作数在主存单元中，而操作数的地址存放在寄存器或另一个主存单元中，指令中给出操作数的地址所在的寄存器编号或主存单元地址。如间接寻址方式、寄存器间接寻址方式。(4)偏移寻址类。指令通过某种方式给出一个形式地址和一个基地址(在某个寄存器中)，经过相应的计算(基地址加形式地址)得到操作数所在单元的地址。有变址、相对和基址寻址3种方式。

16. 直接寻址的操作数需要几次访存？

答：一次。只要根据指令中给出的直接地址进行一次存储访问，取出来的就是操作数。

17. 间接寻址的操作数需要几次访存？

答：至少两次。先根据指令中给出的间接地址进行一次存储访问，取出来的是操作数的地址；再根据操作数的地址访存一次，取出来的才是操作数。所以，一共两次访存。如果是多级间接地址，则要多次访存。

18. 寄存器寻址的操作数需要几次访存？

答：不需要访存。从指定寄存器中取出的就是操作数。

19. 寄存器间接寻址的操作数需要几次访存？

答：一次。先从指令给出的寄存器中取出操作数地址，再根据操作数地址访存一次，得到的就是操作数。

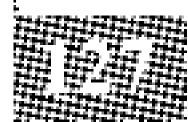
20. 什么是变址寻址方式？

答：变址寻址方式下，指令中的地址码给出一个形式地址，并且隐含或明显地指定一个寄存器作为变址寄存器，变址寄存器的内容(变址值)和形式地址相加，得到操作数的有效地址，根据有效地址进行存储访问，去取操作数或写运算结果。

变址寻址方式的应用很广泛。最基本的使用场合是用在对数组元素的访问中。指令将数组的首地址指定为形式地址，变址寄存器的内容是数组元素的下标，随着下标的变化，可以访问数组中不同的元素。所以变址寄存器的内容是变化的，反映的是所访问的数据到数组首地址的距离，称为变址值。这种应用场合下，形式地址的位数较长，而变址值位数少。变址寻址方式的指令一般包含在一个循环体内。每次进入循环时，变址值都增加或减少一个定长值，这个定长值等于数组元素的长度。

21. 什么是基址寻址方式？

答：基址寻址方式下，指令中的地址码给出一个形式地址，作为位移量，并且隐含或明显地指定一个寄存器作为基址寄存器，基址寄存器的内容和形式地址相加，得到操作数的有效地址，根据有效地址进行访存，去取操作数或写运算结果。



基址寻址的典型应用有两个：一个是程序重定位。在多道程序运行的系统中，每个用户程序在一个逻辑地址空间里编写程序。装入计算机运行时，由操作系统给用户程序分配主存空间，每个用户程序有一个基地址，存放在基址寄存器中，在程序执行时，通过基址寄存器的值加上指令中的形式地址就可以形成实际的主存单元地址。第二个应用是扩展指令的寻址空间。即在运行时将某个主存区间的首地址或程序段的首地址装入基址寄存器，而形式地址给出要访问的单元相对于该首地址的距离(即偏移量)，因此指令中只要用较短的地址码来表示偏移量。访问操作数时，用基址寄存器的值和偏移量相加，得到操作数的有效地址。只要基址寄存器的内容更改到另外一个地址，则操作数的地址空间就移到另一个存储区间。因而可以访问到整个地址空间，以实现短地址访问大空间的目的。

22. 变址寻址方式和基址寻址的区别是什么？

答：变址寻址方式和基址寻址方式的有效地址形成过程类似。但是，基址寻址方式与变址寻址方式在以下方面不同：(1)具体应用的场合不同。变址寻址面向用户，可用于访问字符串、数组、表格等成批数据或其中的某些元素。基址寻址面向系统，用于解决程序的重定位问题和短地址访问大空间的问题。(2)使用方式不同。变址寻址时，指令中提供的形式地址是一个基准地址，位移量由变址寄存器给出；而基址寻址时，指令中给出的形式地址为位移量，而基址寄存器中存放的是基准地址。不过，这里所讲的使用方式并不是绝对的，在实际的计算机设计中，可能会有不同的应用场合和使用方式。

23. 什么是相对寻址方式？

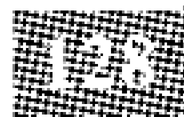
答：指令中的形式地址给出一个位移量D，而基准地址由程序计数器PC提供。位移量给出的是相对于当前指令所在存储单元的距离，位移量可正可负。也就是说，需访问的信息可以在当前指令之前的D个单元处，也可以在当前指令之后的D个单元处。

24. 相对寻址方式用在哪些场合？

答：相对寻址方式用在以下两种场合。(1)公共子程序的浮动。因为公共子程序可能被许多用户程序调用，因而会随着用户程序被装入主存不同的地方运行。为了让公共子程序能在不同的主存区正确运行，一般在公共子程序内部采用相对寻址方式，以保证指令的操作数总在相对于指令的距离一定的单元内。这样，不管子程序浮动到哪里，指令和数据的相对位置不变。例如，现行指令的地址为2000H，指令中给出的形式地址为05H，说明操作数在当前指令后面第05H个单元处，即2005H处。当程序向后浮动了1000H，使当前指令的地址为3000H时，此时公共子程序中的指令、数据以及相对位置都不变，指令中给出的相对地址还是05H，操作数还是应该在当前指令后面的第05H个单元处，所以应该在3005H处，因此，指令取到的还是同一个数据。(2)转移目标地址的寻址。当需要转到当前指令的前面或后面第n条指令执行时，可以用相对寻址方式。此时，得到的转移地址是一个相对地址。

25. 相对寻址方式中如何确定相对位置？

答：相对寻址方式中，相对位置的确定比较复杂。必须注意两个方面的问题：(1)位移量的问题。位移量位数有限，在进行有效地址计算时需要扩展。一般位移量用补码表示，所以应采用补码扩展方式(即符号扩展方式)。(2)基准地址问题。相对寻址的基本思路是把相对于当前指令前面或者后面第n个单元作为操作数或目标转移指令的地址。但在具体实现时，不同机器对“当前指令”的含义有不同的理解。有的机器在计算相对地址时，PC中存



放的还是当前正在执行的指令的地址，但有的机器PC加“1”的操作在取指令时同时完成，所以在计算相对地址时，PC中已经是下一条指令的地址了。因此，不同的机器在计算相对地址时可能有一点细微的差别。

26. 堆栈寻址方式中如何对堆栈进行操作？

答：堆栈(stack)是一块特殊的存储区。采用“先进后出”的方式进行访问。栈底固定不动，栈顶浮动，用一个专门的寄存器(SP)来作为栈顶指针。从堆栈生长的方向来分，可以有“自顶向下”和“自底向上”两种堆栈，它们在进、出栈时对栈指针的修改是不同的。若每个栈中的元素只占一个主存单元，则修改指针时，通过“+1”或“−1”实现；若占多个主存单元，则应该加上或减去相应的值。

假定栈指针指向的总是栈顶处非空元素，则应该按以下方式修改栈指针。对于“自底向上”生成的堆栈，进栈时先修改栈指针：(SP)+1→SP，然后再压入数据；出栈时先将数据弹出，然后再修改栈指针：(SP)−1→SP。对于“自顶向下”生成的堆栈，进栈时先修改栈指针：(SP)−1→SP，然后再压入数据；出栈时先将数据弹出，然后再修改栈指针：(SP)+1→SP。

假定栈指针指向的总是栈顶处的空元素，则应该按以下方式修改栈指针。对于“自底向上”生成的堆栈，进栈时先压入数据，然后再修改栈指针：(SP)+1→SP；出栈时先修改栈指针：(SP)−1→SP，然后再将数据弹出。对于“自顶向下”生成的堆栈，进栈时先压入数据，然后再修改栈指针：(SP)−1→SP；出栈时先修改栈指针：(SP)+1→SP，然后再将数据弹出。

27. 返回指令要不要有地址字段？

答：不一定。子程序的最后一条指令一定是返回指令。一般返回地址保存在堆栈中，所以返回指令中不需要明显给出返回地址，直接从栈顶取地址作为返回地址。如果有些计算机不使用堆栈保存返回地址，而是存放到其他不确定的地方，则返回指令中必须有一个地址码，用来指出返回地址或指出返回地址的存放位置。

28. 转移指令和转子(调用)指令的区别是什么？

答：转移指令有无条件转移指令和条件转移指令(也叫分支指令)。这种转移指令用于改变程序执行的顺序，转移后不再返回来执行，所以无须保存返回地址。而转子指令是一种子程序调用指令，子程序执行结束时，必须返回到转子指令后面的指令执行。所以转子指令执行时，除了和转移指令一样要计算跳转的目标地址外，还要保存返回地址。一般将转子指令后面那条指令的地址作为返回地址保存到一个特殊的寄存器或堆栈中。

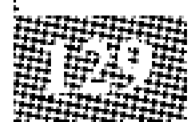
5.5 单项选择题

1. 寄存器中的值有时是地址，有时是数据，它们在形式上没有差别，只有通过(　　)才能识别它是数据还是地址。

   A. 寄存器编号　　　　　　B. 判断程序

   C. 指令操作码或寻址方式位　　D. 时序信号

2. 单地址双目运算类指令中，除地址码指明的一个操作数以外，另一个操作数通常采



用(　　)。

A. 堆栈寻址方式　　B. 立即寻址方式

C. 间接寻址方式　　D. 隐含指定方式

3. 某计算机为定长指令字结构,采用扩展操作码编码方式,指令长度为16位,每个地址码占4位,三地址指令15条,二地址指令8条,一地址指令127条,则剩下的零地址指令最多有(　　)条。

A. 15　　B. 16　　C. 31　　D. 32

4. 在计算机系统中,描述系统运行状态的部件是(　　)。

A. 程序计数器　　B. 累加器

C. 通用寄存器　　D. 程序状态字寄存器

5. 下列有关标志寄存器的叙述中,错误的是(　　)。

A. 可用它来存放执行指令得到的各种标志信息

B. 可通过指令直接访问标志寄存器并修改其值

C. 条件转移指令根据其中的标志位确定PC的值

D. 不需像通用寄存器那样对标志寄存器进行编号

6. 以下给出的4种指令类型中,执行时间最长的指令类型是(　　)。

A. RR型　　B. RS型　　C. SS型　　D. RI型

7. 假定指令地址码给出的是操作数的存储地址,则该操作数采用的是(　　)寻址方式。

A. 立即　　B. 直接　　C. 基址　　D. 相对

8. 假定指令地址码给出的是操作数本身,则该操作数采用的是(　　)寻址方式。

A. 立即　　B. 直接　　C. 基址　　D. 相对

9. 假定指令地址码给出的是操作数所在的寄存器的编号,则该操作数采用的是(　　)寻址方式。

A. 直接　　B. 间接

C. 寄存器直接　　D. 寄存器间接

10. 寄存器间接寻址方式的操作数存放在(　　)中。

A. 通用寄存器　　B. 存储单元

C. 程序计数器　　D. 堆栈

11. 若指令地址码为D,则相对寻址方式下操作数的有效地址为(　　)。

A. D　　B. M[D]　　C. R[D]　　D. PC+D

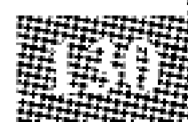
12. 若变址寄存器编号为X,形式地址为D,则变址寻址方式的有效地址为(　　)。

A. R[X]+D　　B. R[X]+R[D]

C. M[R[X]+D]　　D. M[R[X]+M[D]]

13. 假定采用相对寻址方式的转移指令占两个字节,第一字节是操作码,第二字节是相对位移量(用补码表示)。取指令时,每次CPU从存储器取出一个字节,并自动完成PC加1的操作。假设执行到某转移指令时(即取指令前)PC的内容为200CH,该指令的转移目标地址为1FB0H,则该转移指令第二字节的内容应为(　　)。

A. 5CH　　B. 5EH　　C. A2H　　D. A4H



14. 假设某指令的一个操作数采用变址寻址方式，变址寄存器中的值为124，指令中给出的形式地址为B000H，地址B000H中的内容为C000H，则该操作数的有效地址为(　　)。

A. B124H　　B. C124H　　C. B07CH　　D. C07CH

15. 假设某计算机采用小端方式存储，按字节编址。一维数组 a 有100个元素，其类型为float，存放在地址C000 1000H开始的连续区域中，则最后一个数组元素的MSB所在的地址应为(　　)。

A. C000 1396H　　B. C000 1399H
C. C000 118CH　　D. C000 118FH

16. 假设某条指令的一个操作数采用一次间接寻址方式，指令中给出的地址码为1200H，地址1200H中的内容为12FCH，地址12FCH中的内容为38B8H，地址38B8H中的内容为88F9H，则该操作数的有效地址为(　　)。

A. 1200H　　B. 12FCH　　C. 38B8H　　D. 88F9H

17. 假设某条指令的一个操作数采用寄存器间接寻址方式，假定指令中给出的寄存器编号为8，8号寄存器的内容为1200H，地址1200H中的内容为12FCH，地址12FCH中的内容为38B8H，地址38B8H中的内容为88F9H，则该操作数的有效地址为(　　)。

A. 1200H　　B. 12FCH　　C. 38B8H　　D. 88F9H

18. 某计算机按字节编址，采用大端方式存储信息。其中，某指令的一个操作数的机器数为ABCD 00FFH，该操作数采用基址寻址方式，指令中形式地址(用补码表示)为FF00H，当前基址寄存器的内容为C000 0000H，则该操作数的LSB(即FFH)存放的地址是(　　)。

A. C000 FF00H　　B. C000 FF03H
C. BFFF FF00H　　D. BFFF FF03H

19. 某计算机按字节编址，采用小端方式存储信息。其中，某指令的一个操作数为16位，该操作数采用基址寻址方式，指令中形式地址(用补码表示)为FF00H，当前基址寄存器的内容为C000 0000H，则该操作数的LSB存放的地址是(　　)。

A. C000 FF00H　　B. C000 FF01H
C. BFFF FF00H　　D. BFFF FF01H

20. 输入/输出指令的功能是(　　)。

A. 在主存与CPU之间进行数据传送
B. 在主存和I/O端口之间进行数据传送
C. 在CPU和I/O端口之间进行数据传送
D. 在I/O端口和I/O端口之间进行数据传送

21. 通常将在部件之间进行数据传送的指令称为传送指令。以下有关各类传送指令功能的叙述中，错误的是(　　)。

A. 出/入栈指令(Push/Pop)完成CPU和栈顶之间的数据传送
B. 访存指令(Load/Store)完成CPU和存储单元之间的数据传送
C. I/O指令(In/Out)完成CPU和I/O端口之间的数据传送
D. 寄存器传送指令(Move)完成CPU和寄存器之间的数据传送

22. 下列有关 RISC 特征的描述中，错误的是（　　）。

A. 指令格式规整，寻址方式少

B. 采用硬连线控制和指令流水线

C. 配置的通用寄存器数目不多

D. 运算类指令的操作数不访存

23. 假定编译器对 C 源程序中的变量和 MIPS 中寄存器进行了以下对应：变量 f、g、h、i 和 j 分别分配给寄存器 \$s0、\$s1、\$s2、\$s3 和 \$s4，并将一条 C 赋值语句编译后生成如下汇编代码序列：

```
add   $t0,  $s1,  $s2
add   $t1,  $s3,  $s4
sub   $s0,  $t0,  $t1
```

请问这条 C 赋值语句是（　　）。

A. f=(g+i)-(h+j)　　　B. f=(g+j)-(h+i)

C. f=(g+h)-(i+j)　　　D. f=(i+j)-(g+h)

24. 以下有关调用指令（转子指令）的叙述中，错误的是（　　）。

A. 与高级语言源程序中的过程调用相对应，一次过程调用对应一条调用指令

B. 指令执行时必须保留调用指令随后一条指令的地址作为返回地址

C. 递归调用时返回地址通常保存在栈中，非递归调用时可保存在特定寄存器中

D. 指令执行时将无条件转移到目标地址处，转移目标地址无须在指令中明显给出

25. 栈（Stack）是一块采用（　　）方式进行数据存取的存储区，在大多数系统（如 MIPS）中，栈位于高端地址空间，向低地址方向动态增长。

A. 顺序访问　　B. 随机访问　　C. 先进先出　　D. 先进后出

26. 以下有关栈（Stack）和栈帧（Stack Frame）的叙述中，错误的是（　　）。

A. 栈区由若干个栈帧组成，每个栈帧对应一个过程或子程序

B. CPU 中必须有一个专门的栈指针寄存器，用来指示栈顶位置

C. 访存指令不能访问栈中信息，必须提供专门的入栈和出栈指令

D. 过程返回时，应通过修改栈指针寄存器将对应栈帧从栈中退出

【参考答案】

1. C	2. D	3. B	4. D	5. B	6. C	7. B
8. A	9. C	10. B	11. D	12. A	13. C	14. C
15. D	16. B	17. A	18. D	19. C	20. C	21. D
22. C	23. C	24. D	25. D	26. C		

5.6 分析应用题

1. 以下哪种类型的指令执行后一定会改变程序执行顺序？

条件转移指令、无条件转移指令、调用指令、过程返回指令、自陷指令、中断返回指令

【分析解答】

给出的几种指令中，一定会改变程序执行顺序的指令类型有无条件转移指令、过程调用

指令、过程返回指令、自陷指令和中断返回指令。而条件转移指令则在条件不满足时顺序执行指令。

2. 某计算机字长 32 位，CPU 中有 32 个 32 位通用寄存器，采用单字长定长指令字格式，操作码占 6 位，其中还包含对寻址方式的指定。对于存储器直接寻址方式的 RS 型指令，能直接寻址的最大地址空间大小是多少？对于采用通用寄存器作为基址寄存器的 RS 型指令，则能直接寻址的最大地址空间大小是多少？

【分析解答】

因为有 32 个通用寄存器，所以寄存器编号为 5 位。存储器直接寻址的 RS 型指令的一个操作数在寄存器中，所以指令中有一个 5 位的寄存器编号，另外一个地址码是直接地址，共有 32－6－5＝21 位。所以，能直接寻址的最大地址空间大小是 2^{21} 个字。

基址寻址的 RS 型指令的一个操作数在寄存器中，另一个操作数在基址寻址的主存单元中，因为采用通用寄存器作为基址寄存器，所以必须在指令中明显指出基址寄存器是哪个通用寄存器，所以基址寄存器的编号占 5 位，剩下的位数(32－6－5－5＝16 位)是位移量。通用寄存器的位数是 32 位，所以基址寄存器中的基准地址位数是 32 位。一个 32 位的数加上一个 6 位的数，其结果还是一个 32 位的数。所以，能直接寻址的最大地址空间大小是 2^{32} 个字。

3. 若采用相对寻址方式的转移指令占两个字节，第一字节是操作码，第二字节是相对位移量(用补码表示)。CPU 在执行指令时，每次从存储器取出一个字节，并自动完成 PC 加 1。假设某转移指令的地址为 200AH，如果其转移目标地址为 2000H，则该转移指令第二字节的内容应为多少？

【分析解答】

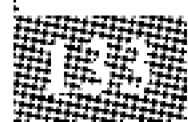
该转移指令的地址为 200AH，因此，在 CPU 取指令过程中，取出第一字节的操作码后，PC 的内容为 200BH，取出第二字节的位移量后，PC 的内容为 200CH，因此，在执行该转移指令计算转移目标地址时，PC 中已经是 200CH 了。因为转移目标地址为 2000H，所以，此时位移量是 2000H－200CH＝－0CH，用补码表示为 100H－0CH＝F4H。

4. 某指令系统的指令字是 16 位，每个地址码为 6 位。若二地址指令 15 条，一地址指令 48 条，则剩下的零地址指令最多有多少条？

【分析解答】

操作码按短到长进行扩展编码。对于二地址指令，两个地址码占 12 位，剩下的操作码占 4 位，最多有 16 种编码，15 条指令用掉 15 种编码 0000～1110，还剩一种编码 1111；对于一地址指令，高 4 位操作码一定是 1111，最低 6 位是一个地址码，剩下的中间操作码还有 6 位，最多可以有 64 种编码，指令条数是 48，因此只需从 64 种编码中选 48 种作为 48 条指令的操作码。可采用如下的操作码编码方案：1111 0 00000～1111 0 11111 (共 32 种编码)、1111 1 0 0000～1111 1 0 1111 (共 16 种编码)；对于零地址指令，其高 10 位操作码的编码空间为 1111 1 1 0000～1111 1 1 1111，因此，高 10 位共有 16 种编码可用，再加上低 6 位的 64 种编码，一共可组合成 16×64＝1024 种编码，可以分别分配给 1024 种指令。故剩下的零地址指令最多有 1024 条。

5. 某计算机指令系统采用定长指令字格式，指令字长 16 位，每个操作数的地址码长 6 位。指令分二地址、一地址和零地址 3 类。若二地址指令有 $k2$ 条，无地址指令有 $k0$ 条，则



一地址指令最多有多少条?

【分析解答】

设一地址指令 $k1$ 条,则 $((16-k2)\times 2^6-k1)\times 2^6=k0$,故 $k1=(16-k2)\times 2^6-k0/2^6$。

6. 假设某计算机按字节编址,采用小端方式存储信息。若某指令的操作数为 32 位字,采用基址寻址方式,指令中形式地址(用补码表示)为 B000H,当前基址寄存器的内容为 8000 4000H,则该指令的操作数地址为多少?若该操作数的机器码为 1234 5678H,则该操作数的 4 个字节 12H、34H、56H 和 78H 的存放地址分别是什么?

【分析解答】

可用两种方法计算操作数地址。方法一:8000 4000H+FFFF B000H=7FFF F000H。

方法二:B000H=1011 0000 0000 0000B,形式地址的值为-101 0000 0000 0000B=-5000H,所以操作数地址为 8000 4000H-5000H=7FFF F000H。小端方式下,LSB 的地址为操作数地址,即书写顺序与存储顺序相反,因而 12H、34H、56H 和 78H 的存放地址分别是 7FFF F003H、7FFF F002H、7FFF F001H 和 7FFF F000H。

7. 一次间接寻址指令中给出的地址码为 2000H,地址为 2000H 的存储单元中的内容为 3000H,地址为 3000H 的存储单元的内容为 4000H,而 4000H 单元的内容为 5000H,则该操作数的有效地址是多少?该操作数的值是多少?

【分析解答】

一次间接寻址方式的指令中给出的地址码是一个间接地址,即操作数地址的地址。所以,操作数的有效地址应该是地址码 2000H 中的内容,即 3000H;有效地址所指出的存储单元的内容是操作数本身,即 4000H 是操作数。

8. 假设地址为 1200H 的存储单元中的内容为 120CH,地址为 120CH 的存储单元的内容为 38B8H,而 38B8H 单元的内容为 88F9H。说明以下各情况下操作数的有效地址和操作数各是多少?

(1) 操作数采用变址寻址,变址寄存器中的值为 12,指令中给出的形式地址为 1200H。

(2) 操作数采用一次间接寻址,指令中给出的地址码为 120CH。

(3) 操作数采用寄存器间接寻址,指令中给出的寄存器编号为 8,8 号寄存器的内容为 1200H。

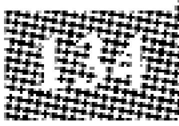
【分析解答】

(1) 有效地址为 000CH+1200H=120CH,操作数为 38B8H。

(2) 有效地址为 38B8H,操作数为 88F9H。

(3) 有效地址为 1200H,操作数为 120CH。

9. 某计算机字长 16 位,存储器存取宽度为 16 位,CPU 中有 8 个 16 位通用寄存器。现为该机设计指令系统,要求指令长度为字长的整数倍,至少支持 64 种不同操作,每个操作数都支持 4 种寻址方式:立即(I)、寄存器直接(R)、寄存器间接(S)和变址(X)寻址方式。存储器地址位数和立即数均为 16 位,任何一个通用寄存器都可作变址寄存器,支持以下 7 种二地址指令格式(R、I、S、X 代表上述 4 种寻址方式):RR 型、RI 型、RS 型、RX 型、XI 型、SI 型、SS 型。请设计该指令系统的 7 种指令格式,给出每种格式的指令长度、各字段所占位数和含义,并说明每种格式指令的功能以及需要的访存次数?



【分析解答】

因为至多有 64 种操作,所以操作码字段只需要 6 位;有 8 个通用寄存器,所以寄存器编号至少占 3 位;寻址方式有 4 种,所以寻址方式位至少占 2 位;直接地址和立即数都是 16 位;任何通用寄存器都可作变址寄存器,所以指令中需明显指定变址寄存器,其编号占 3 位;指令总位数是 16 的倍数。此外,指令格式应尽量规整,指令长度应尽量短。按照上述这些要求设计出的指令格式可以有很多种。以下是采用二地址指令格式的两种指令格式设计方案,RI、XI 和 SI 三种指令格式中添了 3 个 0,是为了补足位数,以使指令长度为 16 的倍数。这两种方案得到的 RR、RS 和 SS 型指令都是 16 位,RI、RX 和 SI 型指令都是 32 位,XI 型指令是 48 位。

指令格式示例 1：如图 5.1 所示,其中最左边的 4 位为"类型"字段,用于说明不同的指令类型,这样,不用对两个操作数的寻址方式分别说明。7 种指令类型只要 3 位编码即可,所以最后 1 位总是"0"。

类型						
RR型	0000	OP(6位)	Rt (3位)	Rs (3位)		
RI型	0010	OP(6位)	Rt (3位)	000	Imm16(16位)	
RS型	0100	OP(6位)	Rt (3位)	Rs (3位)		
RX型	0110	OP(6位)	Rt (3位)	Rx (3位)	Offset16(16位)	
XI型	1000	OP(6位)	Rx (3位)	000	Offset16(16位)	Imm16(16位)
SI型	1010	OP(6位)	Rt (3位)	000	Imm16(16位)	
SS型	1100	OP(6位)	Rt (3位)	Rs (3位)		

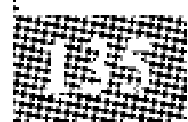
图 5.1　第一种指令格式示例

指令格式示例 2：如图 5.2 所示,用专门的"寻址方式"字段分别说明两个操作数的寻址方式。其定义为 00-立即、01-寄存器直接、10-寄存器间接、11-变址。

类型							
RR型	OP(6位)	01	01	Rt (3位)	Rs (3位)		
RI型	OP(6位)	01	00	Rt (3位)	000	Imm16(16位)	
RS型	OP(6位)	01	10	Rt (3位)	Rs (3位)		
RX型	OP(6位)	01	11	Rt (3位)	Rx (3位)	Offset16(16位)	
XI型	OP(6位)	11	00	Rx (3位)	000	Offset16(16位)	Imm16(16位)
SI型	OP(6位)	10	00	Rt (3位)	000	Imm16(16位)	
SS型	OP(6位)	10	10	Rt (3位)	Rs (3位)		

图 5.2　第二种指令格式示例

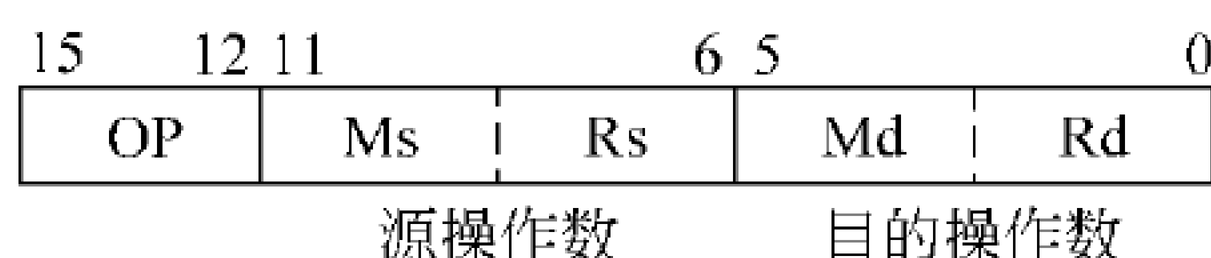
存储器存取宽度为 16 位,说明每次可从存储器取出 16 位。因此,读取 16、32 和 48 位指令分别需要 1、2 和 3 次存储器访问。各类指令的功能和访存次数分别说明如下(M[x]表



示存储器地址 x 中的内容，R[x]表示寄存器 x 中的内容)。

RR 型指令的功能为 R[Rt]←R[Rt] op R[Rs]，访存 1 次；RI 型指令的功能为 R[Rt]←R[Rt] op Imm16，访存 2 次；RS 型指令的功能为 R[Rt]←R[Rt] op M[R[Rs]]，访存 2 次；RX 型指令的功能为 R[Rt]←R[Rt] op M[R[Rx]+Offset]，访存 3 次；XI 型指令的功能为 M[R[Rx]+Offset]←M[R[Rx]+Offset] op Imm16，访存 5 次；SI 型指令的功能为 M[R[Rt]]←M[R[Rt]] op Imm16，访存 4 次；SS 型指令的功能为 M[R[Rt]]←M[R[Rt]] op M[R[Rs]]，访存 4 次。(上述给出的访存次数中包括读取指令过程中的访存次数)

10. 某计算机字长为 16 位，主存地址空间大小为 128 KB，按字编址。采用单字长定长指令格式，指令各字段定义如下：



转移指令采用相对寻址方式，相对偏移量用补码表示。寻址方式定义如表 5.3 所示。

表 5.3　题 10 中定义的寻址方式及其含义

Ms/Md	寻 址 方 式	助记符	含　　义
000B	寄存器直接	Rn	操作数=R[Rn]
001B	寄存器间接	(Rn)	操作数=M[R[Rn]]
010B	寄存器间接、自增	(Rn)+	操作数=M[R[Rn]]，R[Rn]←R[Rn]+1
011B	相对	D(Rn)	转移目标地址=PC+R[Rn]

(注：M[x]表示存储器地址 x 中的内容，R[x]表示寄存器 x 中的内容)

请回答下列问题：

(1) 该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器(MAR)和存储器数据寄存器(MDR)至少各需要多少位？

(2) 转移指令的目标地址范围是多少？

(3) 若操作码 0010B 表示加法操作(助记符为 add)，寄存器 R4 和 R5 的编号分别为 100B 和 101B，R4 的内容为 1234H，R5 的内容为 5678H，地址 1234H 中的内容为 5678H，地址 5678H 中的内容为 1234H，则汇编语句“add (R4)，(R5)+”(逗号前为第二源操作数，逗号后为第一源操作数和目的操作数)对应的机器码是什么(用十六进制表示)？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？

【分析解答】

(1) 因为采用单字长指令格式，操作码字段占 4 位，所以最多有 16 条指令；指令中通用寄存器编号占 3 位，所以，最多有 8 个通用寄存器；因为主存地址空间大小为 128KB，按字编址，故共有 64K 个存储单元，因而地址位数为 16 位，所以 MAR 至少为 16 位；因为字长为 16 位，所以 MDR 至少为 16 位。

(2) 因为地址位数和字长都为 16 位，所以 PC 和通用寄存器位数均为 16 位，两个 16 位数据相加其结果也为 16 位，即转移目标地址位数为 16 位，因而能在整个地址空间转移，即目标转移地址的范围为 0000H～FFFFH。

(3) 要得到汇编语句“add (R4),(R5)+”对应的机器码，只要将其对应的指令代码各个字段拼接起来即可。显然，add 对应 OP 字段，为 0010B；(R4)的寻址方式字段为 001B，R4 的编号为 100B；(R5)+的寻址方式字段为 010B，R5 的编号为 101B；所以，对应的机器码为 0010 001 100 010 101B，用十六进制表示为 2315H。指令“add (R4)，(R5)+”的功能为 M[R[R5]]←M[R[R5]]+M[R[R4]]，R[R5]←R[R5]+1。已知 R[R4]=1234H，R[R5]=5678H，M[1234H]=5678H，M[5678H]=1234H，因为 1234H+5678H=68ACH，所以 5678H 单元中的内容从 1234H 改变为 68ACH，同时 R5 中的内容从 5678H 变为 5679H。

11. 假定 A 是一个 32 位的地址，A_upper 和 A_lower 分别表示地址 A 的高、低 16 位，以下 MIPS 指令代码用来将存放在存储器地址 A 处的机器码读入寄存器 $ s0 中。

```
lui    $t0,    A_upper             #将 A_upper 的低位添加 16 个 0,送$t0
ori    $t0,    $t0,  A_lower       #将 A_lower 的高位扩展 0 后与$t0 的内容相"或",送$t0
lw     $s0,    0($t0)              #将$t0 的内容和 0 相加得到有效地址,从中取数送$s0
```

上述功能也能用以下两条 MIPS 指令来实现。请问第一条指令中 A_upper_adjusted 的值如何得到？

```
lui  $t0,  A_upper_adjusted
lw  $s0,  A_lower($t0)             #将 A_lower 进行符号扩展后,和$t0 的内容相加,得到有效
                                   #地址,从中取数送$s0
```

【分析解答】

因为 MIPS 指令中的立即数只能是 16 位，所以一个 32 位的地址无法直接传送到一个 32 位寄存器中，第一种方案是通过将地址 A 的高 16 位和低 16 位分别作为两条指令的立即数，将它们用“或”操作合并到一个 32 位寄存器中。这样，第 3 条取数指令 lw 的偏移量就是 0。第二种方案中取数指令 lw 的偏移量是 A 的低位部分 A_lower，由于取数指令 lw 在计算内存单元地址时对偏移量采用的是符号扩展，所以要使得高 16 位最终的结果为 A_upper，必须对 A_upper 进行以下调整：若 A_lower 的最高位(看成低 16 位数的符号位)是 0，则 A_upper_adjusted=A_upper，这样，A_lower 符号扩展后高 16 位为全 0，和高 16 位 A_upper 相加后，高 16 位还是 A_upper；若 A_lower 的最高位是 1，则 A_lower 符号扩展后高 16 位为全 1，此时，A_upper_adjusted 应满足 FFFFH+A_upper_adjusted=A_upper。而因为 FFFFH+A_upper+1=A_upper(最高位向前面的进位被丢弃)，所以，A_upper_adjusted=A_upper+1。

12. 除了硬件乘法器外，还可以用移位和加法指令来实现乘法运算。在乘以较小的常数时，这种办法很有效。在不考虑溢出的情况下，假设要将 $ s0 的内容与 6 相乘，乘积存入 $ s1 中。请写出一段指令条数最少且不包括乘法指令的 MIPS 代码。

【分析解答】

一个数 x 乘以 6，相当于 $4x+2x$，而 $4x$ 可以通过将 x 左移两位来实现，$2x$ 可以通过将 x 左移一位来实现。这样就只要用两条左移指令和一条加法指令来实现乘 6 操作。以此类推，当乘以一个较小的常数时，只要将这个较小的常数分解成若干个 2 的幂次相加，就可以用若干条左移指令和一条加法指令来实现乘法运算。可用以下指令序列实现题目要求。

```
sll    $s1,  $s0,  2            #将$s0的内容左移两位,送$s1
sll    $s0,  $s0,  1            # 将$s0的内容左移一位,送$s0
addu   $s1,  $s1,  $s0          #将$s1和$s0的内容相加(不考虑溢出),送$s1
```

13. 有些计算机提供了专门的指令,能从32位寄存器中抽取其中任意一个位串置于另一个寄存器的低位有效位上,并高位补0,如图5.3所示。MIPS指令系统中没有这样的指令,请写出最短的一个MIPS指令序列来实现这个功能,要求 $i=5$, $j=22$,操作前后的寄存器分别为 \$s0 和 \$s2。

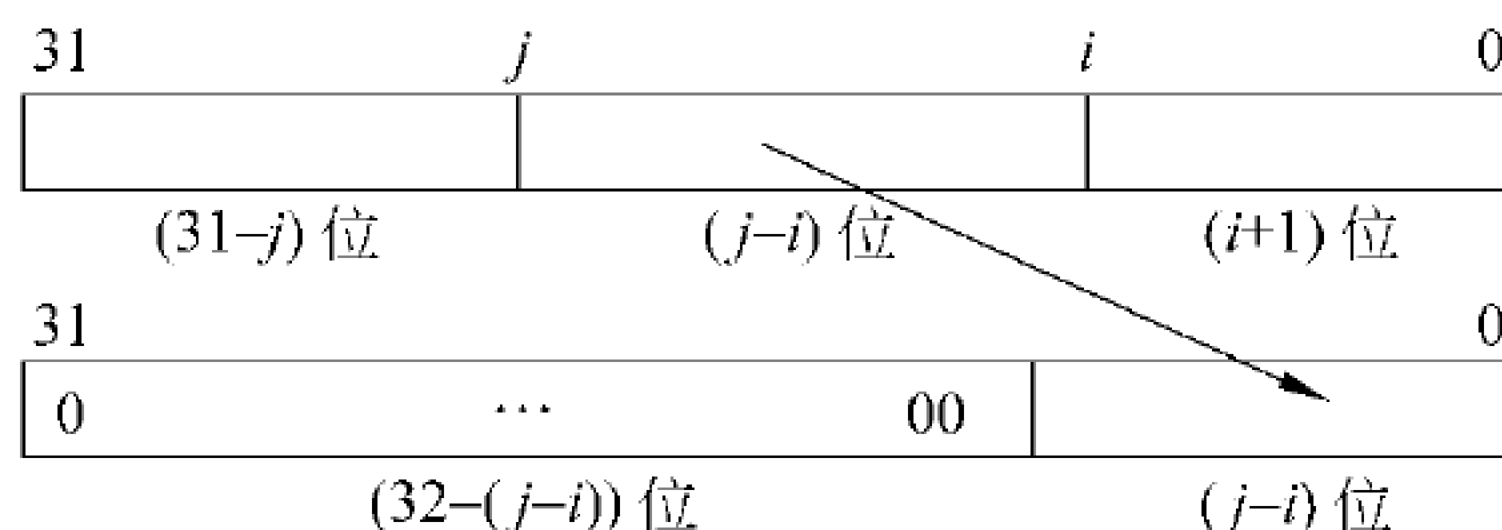


图5.3　题13中操作前后示意图

【分析解答】

可以先左移9位,然后右移15位,MIPS指令序列为:

```
sll  $s2,  $s0,  9             #将$s0的内容左移9位,送$s2
srl  $s2,  $s2,  15            #将$s2的内容右移15位,送$s2
```

这里要注意,第二条指令不能用算术右移指令sra,因为算术右移高位添加的是符号,所以,不能保证高位一定补0。此外,第一条指令中的目的操作数寄存器和第二条指令的源操作数寄存器都只能用 \$s2,而不能改成其他寄存器,否则,会破坏其他寄存器的值!

14. 以下程序段是某个过程对应的指令序列。入口参数 int a 和 int b 分别置于 \$a0 和 \$a1 中,返回参数是该过程的结果,置于 \$v0 中。要求为以下MIPS指令序列加注释,并简单说明该过程的功能。

```
          add    $t0,  $zero,  $zero
loop:     beq    $a1,  $zero,  finish
          add    $t0,  $t0,   $a0
          sub    $a1,  $a1,   1
          j      loop
finish:   addi   $t0,  $t0,   100
          add    $v0,  $t0,   $zero
```

【分析解答】

```
          add    $t0,  $zero,  $zero      #将寄存器$t0置0
loop:     beq    $a1,  $zero,  finish     #若$a1的值等于0,则转finish处
          add    $t0,  $t0,   $a0         #将$t0和$a0的内容相加,送$t0
          sub    $a1,  $a1,   1           #将$a1的值减1
          j      loop                     #无条件转到loop处
finish:   addi   $t0,  $t0,   100         #将$t0的内容加100
          add    $v0,  $t0,   $zero       #将$t0的内容送$v0
```

该过程的功能是计算“$a \times b+100$”。

15. 用一条 MIPS 指令或最短的指令序列实现以下 C 语言语句：$b=25|a$。假定编译器将 a 和 b 分别分配到 \$t0 和 \$t1 中。如果把 25 换成 65536，即 $b=65536|a$，则用 MIPS 指令或指令序列如何实现？

【分析解答】

只要用一条指令“ori \$t1，\$t0，25”就可实现 $b=25|a$。但是，如果把 25 换成 65536，则不能用一条指令“ori \$t1，\$t0，65536”来实现，因为 65536＝1 0000 0000 0000 0000B，它不能用 16 位立即数表示。可用以下两条指令实现 $b=65536|a$。

```
lui  $t1,  1                #将 0001 0000H 置于寄存器$t1
or   $t1,  $t0,  $t1        #将$t0 和$t1 的内容进行"或"运算,送$t1
```

16. 请说明 beq 指令的跳转范围，并解释为什么汇编程序在对下列 MIPS 汇编源程序中的 beq 指令进行汇编时会遇到问题，应该如何修改该程序段？

```
here:   beq  $s0,  $s2,  there
        …
there:  addi  $s1,  $s0,  4
```

【分析解答】

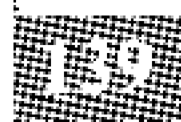
在 MIPS 指令系统中，分支指令 beq 是 I-型指令，转移目标地址采用相对寻址方式，16 位偏移量 offset 用补码表示，因此可以跳转到当前指令前，也可以跳转到当前指令后。其转移目标地址的计算公式为 PC＋4＋Offset×4。因此，当 Offset 的范围为 0000 0000 0000 0000B～0111 1111 1111 1111B 时，beq 指令向后正跳，相对于本条指令，向后正跳 $1\sim 2^{15}$ 条指令，其跳转地址范围为 $4 \sim 4+(2^{15}-1)\times 4=2^{17}$。当 Offset 的范围为 1000 0000 0000 0000B ～ 1111 1111 1111 1111B 时，beq 指令向前负跳，相对于本条指令，向前负跳 $1\sim 2^{15}-1$ 条指令，其跳转地址范围为 $-4\times(2^{15}-1)\sim -4$。

当上述指令序列中 here 和 there 表示的地址相差超过上述给定范围时，beq 指令会发生汇编错误。可将上述指令序列改成以下指令序列。因为无条件跳转指令 j 的跳转范围足够大，所以可以直接从 here 跳转到 there。

```
here:    bne  $s0,  $s2,  skip
          j  there
skip:    …
          …
there:    addi  $s1,  $s0,  4
```

17. 下列指令序列对应一个完整的过程，用来对两个数组进行处理，并产生结果存放在 \$v0 中。假定每个数组有 2500 个元素，每个数组元素都为 int 类型。两个数组的首地址分别存放在 \$a0 和 \$a1 中，数组长度分别存放在 \$a2 和 \$a3 中。根据注释简单说明该过程的功能。假定该过程运行在一个时钟频率为 2GHz 的处理器上，add、addi 和 sll 指令的 CPI 为 1；lw 和 bne 指令的 CPI 为 2，则最坏情况下运行该过程所需要的时间是多少秒？

```
        sll   $a2,  $a2,  2          #$a2 的内容左移 2 位,即乘 4
        sll   $a3,  $a3,  2          #$a3 的内容左移 2 位,即乘 4
```



```
        add   $v0,  $zero,  $zero        #$v0初始化为0
        add   $t0,  $zero,  $zero        #$t0初始化为0
outer:  add   $t4,  $a0,  $t0            #计算数组1当前元素的地址
        lw    $t4,  0($t4)               #数组1当前元素存放在$t4
        add   $t1,  $zero,  $zero        #$t1初始化为0
inner:  add   $t3,  $a1,  $t1            #计算数组2当前元素的地址
        lw    $t3,  0($t3)               #数组2当前元素存放在$t3
        bne   $t3,  $t4,  skip           #若$t3和$t4的内容不相等,则转skip
        addi  $v0,  $v0,  1              #$v0加1
skip:   addi  $t1,  $t1,  4              #$t1加4
        bne   $t1,  $a3,  inner          #数组2未处理完,继续转inner执行
        addi  $t0,  $t0,  4              #$t0加4
        bne   $t0,  $a2,  outer          #数组1未处理完,继续转outer执行
```

【分析解答】

该过程的功能是统计两个数组中相同元素的个数,多次相等则重复计数。最坏的情况是两个数组的所有元素都相等,这样,指令"addi \$v0, \$v0, 1"在每次循环中都被执行。因为add、addi和sll指令的CPI为1,lw和bne指令的CPI为2,所以,在最坏情况下所需要的时钟周期总数为$\{4+[4+(6+3)\times2500+3]\times2500\}=56267506$,时钟周期为1/2GHz=0.5ns,因此,执行该过程的总执行时间最多为$56267506\times0.5\text{ns}=28133753\text{ns}\approx0.028\text{s}$。

18. 以下程序段是某个过程对应的MIPS指令序列,其功能为复制一个存储块数据到另一个存储块中,存储块中每个数据的类型为float,源数据块和目的数据块的首地址分别存放在\$a0和\$a1中,复制的数据个数存放在\$v0中,作为返回参数返回给调用过程。在复制过程中遇到0则停止,最后一个0也需要复制,但不被计数。已知该程序段中有多个Bug,请找出它们并修改。

```
        addi  $v0,  $zero,  0
loop:   lw    $v1,  0($a0)
        sw    $v1,  0($a1)
        addi  $a0,  $a0,  4
        addi  $a1,  $a1,  4
        beq   $v1,  $zero,  loop
```

【分析解答】

修改后的代码如下:

```
        addi  $v0,  $zero,  0
loop:   lw    $v1,  0($a0)
        sw    $v1,  0($a1)
        beq   $v1,  $zero,  exit
        addi  $a0,  $a0,  4
        addi  $a1,  $a1,  4
        addi  $v0,  $v0,  1
        j     loop
exit:
```

19. 考虑下面的C语言程序段：

```
for  (i=0; i<=100; i=i+1; )
     a[i]=b[i]+c;
```

假定数组 *a* 和 *b* 的每个元素都是int型变量，首地址分别存放在寄存器 $ a0 和 $ a1 中。寄存器 $ t0 和 $ s0 分别对应变量 *i* 和 *c*。要求写出与之对应的MIPS指令代码，并计算这段代码运行过程中所执行的指令条数和数据的访存次数？

【分析解答】

每个数组元素占4个字节，所以循环体内每一步地址增量是4。上述程序段对应的MIPS汇编形式的指令代码序列如下。

```
        add   $t0,  $zero,  $zero       #i=0
loop:   add   $t4,  $a1,  $t0           #$t4=address of b[i]
        lw    $t5,  0($t4)              #$t5=b[i]
        add   $t6,  $t5,  $s0           #$t6=b[i]+c
        add   $t7,  $a0,  $t0           #$t7=address of a[i]
        sw    $t6,  0($t7)              #a[i]=b[i]+c
        addi  $t0,  $t0,  4             #i=i+4
        slti  $t8,  $t0,  401           #if (i<401) then $t8=1 else $t8=0
        bne   $t8,  $zero,  loop        #if ($t8=1) go to loop
```

该段代码运行过程中共执行了 1＋101× 8＝809 条指令。其中，访存指令条数为 2×101＝202 条，因此数据的访存次数为 202 次。

20. 某高级语言源程序中的一个while语句为“while(save[i]＝＝k)i＋＝1;”，若对其编译时，编译器将 *i* 和 *k* 分别分配在寄存器 $ s3 和 $ s5 中，数组 *save* 的基址存放在 $ s6 中，则生成的MIPS汇编代码段如下。

```
loop:   sll   $t1, $s3, 2     #R[$t1]←R[$s3]<<2,即 R[$t1]=i×4
        add   $t1, $t1, $s6   #R[$t1]←R[$t1]+R[$s6],即 R[$t1]=Address of save[i]
        lw    $t0, 0($t1)     #R[$t0]←M[R[$t1]+0],即 R[$t0]=save[i]
        bne   $t0, $s5, exit  #if R[$t0]≠ R[$s5] then goto exit
        addi  $s3, $s3, 1     #R[$s3]←R[$s3]+1,即 i=i+1
        j     loop            #goto loop
exit:
```

假设从loop处开始的指令序列存放在内存80000处，则上述循环对应的MIPS机器码如图5.4所示。

根据上述叙述，回答下列问题，要求说明理由或给出计算过程。

(1) MIPS的编址单位是多少？数组 *save* 每个元素占几个字节？

(2) 为什么指令“sll $ t1, $ s3, 2”能实现 4×*i* 的功能？

(3) 该循环指令序列中哪些是R-型指令？哪些是I-型指令？

(4) $ t0 和 $ s6 的编号各为多少？

(5) 指令“j loop”的操作码是什么(用二进制表示)？

(6) 标号exit的值是多少？如何根据指令计算得到？

	6位	5位	5位	5位	5位	6位
80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	21	1		
80020	2	20000				
80024	…					

图 5.4　题 20 中程序的机器级代码

(7) 标号 loop 的值是多少？如何根据指令计算得到？MIPS 中跳转指令的跳转范围是多少？

【分析解答】

(1) MIPS 的编址单位是字节。从图 5.4 中可看出，每条指令 32 位，占 4 个地址，所以一个地址中有 8 位。因为每次循环取数组元素时，其下标地址都要乘以 4，所以 save 数组的每个元素占 4 个字节。

(2) 因为这是左移指令，左移 2 位，相当于乘以 $2^2=4$。

(3) 从图 5.4 可以看出，第 1 和第 2 条为 R-型指令，第 3、4、5 条为 I-型指令。

(4) 从图 5.4 中第 3 和第 4 条指令可看出，$ t0 的编号为 8，从第 2 条指令可看出 $ s6 的编号为 22。

(5) 指令“j loop”的操作码为“000010B”。

(6) 标号 exit 的值是 80024，其含义是循环结束时跳出循环后执行的首条指令的地址。它由当前分支指令(条件转移指令)的地址 80012 加上 4 得到下条指令的地址，然后再加上相对位移量 2×4 得到，即 80012+4+2×4=80024。

(7) 标号 loop 的值为 80000，是循环入口处首条指令的地址，由跳转指令的 32 位地址 80020 的高 4 位(0000B)，与指令中给出的低 26 位(20000)拼接成 30 位地址，然后再在低位添两个 0(相当于×4)得到，即 20000×4=80000。因为跳转指令的地址与其跳转到的目标指令地址的高 4 位一样，所以，如果将 4GB 的主存空间分割成 16 个 256MB 的子空间，那么跳转到的目标指令总是和跳转指令在同一个子空间，不可能跳出它本身所在的 256MB 的子空间，所以跳转目标地址范围的大小是 256M，即假定跳转指令地址的高 4 位为 X，则跳转目标地址范围是 X000 0000H～XFFF FFFCH。

21. 以下 C 语言程序段中有两个过程 sum_array 和 compare，假定 sum_array 第一个被调用，全局变量 sum 分配在寄存器 $ s0 中。要求写出每个过程对应的 MIPS 汇编表示，并说明每个过程对应的栈帧中需要保存的信息。

```
1    int sum=0;
2    int sum_array(int array[], int num)
3    {
4        int i;
5        for (i=0; i <num; i++)
6            if compare (num, i+1)   sum+=array[i] ;
7        return sum;
```

```
8    }
9    int compare (int a, int b)
10   {
11       if (a >b)
12           return 1;
13       else
14           return 0;
15   }
```

【分析解答】

程序由两个过程组成，全局静态变量 sum 分配给 $ s0，在过程调用时，全局变量无须入栈保存。

为了尽量减少指令条数，并减少访存次数，在每个过程的过程体中总是先使用临时寄存器 $ t0～$ t9，临时寄存器不够或者某个值在调用过程返回后还需要用，就使用保存寄存器 $ s0～$ s7。

MIPS 指令系统中没有寄存器传送指令，为了提高汇编表示的可读性，引入一条伪指令 move 来表示寄存器传送，汇编器将其转换为具有相同功能的机器指令。伪指令“move $ t0，$ s0”对应的机器指令为“add $ t0，$ zero，$ s0”。

(1) 过程 sum_array：过程中使用的 *array* 数组在其他过程中已经定义，而不是局部变量，无须在过程的栈帧中给它分配空间，只需将其首地址作为入口参数传递给过程 sum_array(假定在 $ a0 中)。此外，还有另一个入口参数为 *num*(假定在 $ a1 中)，最后有一个返回参数 sum。该过程又调用了 compare 过程，因此它不是叶子过程。所以，其栈帧中除了保留所用的保存寄存器外，还必须保留返回地址 $ ra，以免在调用过程 compare 时被覆盖。是否保存 $ fp 要看具体情况，如果确保后面一直都用不到 $ fp，则可以不保存它，但编译器往往无法预测这种情况，所以通常采用统一的做法，即总是保存 $ fp 的值。因而，该过程的栈帧中要保存的信息有返回地址 $ ra 和帧指针 $ fp，其栈帧空间大小为 4×2=8B。

汇编表示如下：

```
            move  $s0,  $zero          #sum=0
sum_array:  addi  $sp,  $sp,  -8       #generate stack frame
            sw    $ra,  4($sp)         #save $ra on stack
            sw    $fp,  0($sp)         #save $fp on stack
            addi  $fp,  $sp,  4        #set $fp
            move  $t2,  $a0            #base address of array
            move  $t0,  $a1            #$t0=num
            move  $t3,  $zero          #i=0
loop:       slt   $t1,  $t3,  $t0      #if (i<num) $t1=1 else $t1=0
            beq   $t1,  $zero,  exit1  #if ($t1=0) jump to exit1
            move  $a0,  $t0            #$a0=num
            move  $a1,  $t3            #$a1=i
            addi  $a1,  $a1,  1        #$a1=i+1
            jal   compare              #call compare
            beq   $v0,  $zero,  else   #if ($v0=0) jump to else
```

```
            sll     $t1,  $t3,  2          #i=i×4
            add     $t1,  $t2,  $t1        #$t1=array[i]
            lw      $t4,  0($t1)           #load array[i]
            add     $s0,  $s0,  $t4        #sum+=array[i]
else:       addi    $t3,  $t3,  1          #i=i+1
            j       loop
exit1:      move    $v0,$s0                #return sum
            lw      $ra,  4($sp)           #restore $ra
            lw      $fp,  0($sp)           #restore $fp
            addi    $sp,  $sp,  8          #free stack frame
            jr      $ra                    #return to caller
```

(2) 过程 compare：入口参数为 a 和 b，分别在 $ a0 和 $ a1 中，有一个返回参数，没有局部变量，是叶子过程，且过程体中没有用到任何保存寄存器，所以栈帧中不需要保留任何信息。

```
compare:    move    $v0,  $zero            #起初返回值设定为 0
            beq     $a0,  $a1,  exit2      #if ($a0=$a1) jump to exit2
            slt     $t1,  $a0,  $a1        #if ($a0<$a1) $t1=1else $t1=0
            bne     $t1,  $zero,  exit2    #if ($a0<$a1) jump to exit2
            ori     $v0,  $zero, 1         #返回值设定为 1
exit2:      jr      $ra
```

22. 以下是一个计算阶乘的 C 语言递归过程，请按照 MIPS 过程调用协议写出该递归过程对应的 MIPS 汇编语言程序，要求目标代码尽量短(提示：乘法运算可用乘法指令"mul rd, rs, rt"来实现)。

```
int  fact (int n)
{
    if  (n <1)
        return (1) ;
    else
        return (n* fact (n-1) );
}
```

【分析解答】

过程 fact 有一个输入参数 n，按 MIPS 过程调用协议，n 应在 $ a0 中，返回参数应存放在 $ v0 中。过程内没有局部变量，故无须在其栈帧中保留局部变量所用空间；需递归调用 fact 过程，所以必须在其栈帧中保留返回地址 $ ra。过程体内全部使用临时寄存器 $ t0～$ t9，因而无须在其栈帧中保存通用寄存器。因为是递归调用，所以需在栈帧中保留输入参数。因此，该过程的栈帧中要保存的信息有返回地址 $ ra、帧指针 $ fp 和输入参数 $ a0，其栈帧空间大小为 4×3=12B。

```
fact:       addi    $sp,  $sp,  -12        #generate stack frame
            sw      $ra,  8($sp)           #save $ra on stack
            sw      $fp,  4($sp)           #save $fp on stack
```

```
          addi   $fp,  $sp,  0           #set $fp
          sw     $a0,  0($fp)            #save $a0(n)on stack
          slti   $t0,  $a0,  1           #if (n<1) $t0=1 else $t0=0
          bne    $t0,  $zero,  exit1     #if (n<1) goto exit1
          addi   $a0,  $a0,-1            #n=n-1
          jal    fact                    #call fact
          lw     $t1,  0($fp)            #restore n
          mul    $v0,  $t1,  $v0         #$v0=n* fact (n-1)
          j      exit
exit1:    addi   $v0,  $zero,  1         #$v0=1
exit:     lw     $ra,  8($sp)            #restore $ra
          lw     $fp,  4($sp)            #restore $fp
          addi   $sp,  $sp,  12          #free stack frame
          jr     $ra                     #return to caller
```

第 6 章 中央处理器

6.1 教学目标和内容安排

主要教学目标：

使学生了解 CPU 的主要功能、CPU 的内部结构、指令的执行过程、数据通路的基本组成、数据通路的定时、数据通路中信息的流动过程、控制器的实现方式、硬连线路控制器的设计、微程序控制器的设计、异常和中断的概念等，为进一步深入理解流水线 CPU 的设计原理和高级流水线技术打下基础。

基本学习要求：

(1) 了解 CPU 的主要功能。

(2) 了解 CPU 的基本结构。

(3) 理解 CPU 中通用寄存器和专用寄存器的作用。

(4) 了解一条指令的基本执行过程。

(5) 理解指令周期、机器周期、时钟周期的概念及其相互关系。

(6) 了解数据通路的基本组成。

(7) 了解数据通路中哪些是组合逻辑部件，哪些是时序逻辑部件。

(8) 了解数据通路中的组合逻辑部件和时序逻辑部件的差别。

(9) 了解寄存器堆的作用与工作原理。

(10) 了解多路选择器的作用与工作原理。

(11) 了解 ALU 在数据通路中的功能。

(12) 了解加法器和 ALU 的差别。

(13) 了解指令存储器和数据存储器之间的差别。

(14) 了解取指阶段的数据流动过程。

(15) 了解寄存器取数过程。

(16) 了解数据运算过程。

(17) 了解存储器取数时数据流动过程。

(18) 了解寄存器存数时数据流动过程。

(19) 了解如何实现条件转移的数据通路。

(20) 了解如何实现无条件转移的数据通路。

(21) 了解"0"扩展和"符号"扩展的含义和实现方式。

(22) 理解如何确定单周期数据通路的时钟周期。

(23) 理解如何确定多周期数据通路中的时钟周期。

(24) 单周期数据通路和多周期数据通路的差别。

(25) 理解为什么很少有机器采用单周期数据通路。

(26) 理解数据通路的设计和 CPI 之间的关系。

(27) 理解指令格式的规整性对数据通路设计的影响。

(28) 理解各个控制信号的含义、控制点,以及在各指令中的取值。

(29) 了解控制器的设计步骤和实现方式。

(30) 掌握如何用组合逻辑设计方式实现硬布线控制器。

(31) 了解利用微程序设计方式实现微程序控制器的基本原理。

(32) 理解内部异常和外部中断的概念。

(33) 理解为什么在设计处理器时必须考虑异常和中断的处理。

(34) 了解如何在数据通路设计中考虑异常和中断的处理。

(35) 理解内部异常和外部中断的区别。

(36) 理解常见异常事件的含义和处理方式。

(37) 了解带中断的指令执行过程。

本章应该是本课程的核心内容,主要介绍 CPU 中执行指令的数据通路及其控制逻辑电路的设计。其重点内容包括数据通路的定时、单周期数据通路、单周期控制器、微程序概念和带异常和中断处理的处理器实现。

主教材[①]分 CPU 概述、单周期处理器设计、多周期处理器设计、微程序控制器设计、异常和中断处理这 5 个部分进行了介绍。

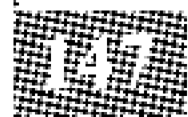
在第一部分 CPU 概述中,给出了 CPU 设计涉及的基本问题、基本概念和 CPU 设计的基本思路。这部分应该是最基础的内容,学生应该很好地掌握。不过,对其中一些概念和知识的理解,还需要在后面具体的数据通路设计和控制器设计的学习过程中得到深化。因此,在后面单周期处理器设计和多周期处理器设计的内容介绍过程中,可以通过对具体情况的分析,来强化 CPU 概述中介绍的内容。

为了全面清楚地说明 CPU 数据通路结构的发展过程,主教材在 CPU 概述中简单介绍了早期累加器型指令系统的数据通路、单总线数据通路和三总线数据通路,与后面介绍的单周期数据通路、多周期数据通路,以及第 7 章介绍的流水线数据通路和高级流水线数据通路串接起来,就形成了数据通路结构发展演变的技术路线图,建议教学过程中要有意识地帮助学生理解这个技术演变过程,并分析这种演变过程的原因所在。在这部分内容中,可对有关单总线数据通路的内容进行较为详细的介绍,而对早期累加器型指令系统的数据通路和三总线数据通路简单说明一下即可。

在介绍单周期处理器和多周期处理器设计内容时,主要以 MIPS 指令系统中有代表性的几条指令作为实现目标。其中,对单周期处理器设计内容介绍较为详细,这样做的原因有

---

① 主教材指《计算机组成与系统结构》(袁春风编著,清华大学出版社,2010.4)



两个，第一，因为单周期处理器的结构与流水线处理器的结构比较类似，掌握单周期数据通路及其控制逻辑电路的设计方法，能更好地理解流水线处理器的设计方法。第二，因为单周期处理器设计过程比较简单，便于学生理解 CPU 设计的原理性内容。因此，建议在课堂教学中对单周期处理器的设计内容进行较为详细的介绍。在课时有限的情况下，对于多周期处理器的设计，就无须详细展开讲解，只要简单说明一下基本设计思想和基本实现原理即可。

对于微程序控制器设计，着重讲清楚微程序控制器的基本设计思想和基本结构、微指令格式和微命令编码方式，以及微程序执行顺序的控制方式。主教材中例 6.2、例 6.3 和例 6.4，都是为了便于理解基本原理而给出的，主要是为了给学生提供具体示例，以达到通过对概念的具体运用来更好地理解概念的目的。对于这些例子，在课时有限的情况下，课堂上只要大致讲一下字段如何划分，然后对其中的一个字段简单说明一下如何编码即可，不需要在课堂上详细展开讲解，细节问题都可留给学生自学，如果学生自学时不能明白一些具体的细节问题，也没有关系，只要学生通过例子能够掌握基本原理就行了。

本章最后一个内容是异常和中断处理，应是本课程和操作系统课程中最重要的概念之一，对学生将来从事处理器设计、操作系统开发和设计、嵌入式软硬件设计等都非常有用。对于这部分内容，学生普遍存在的一些问题是：分不清异常和中断在检测、响应和处理过程中的不同；分不清 CPU 和I/O 接口中各自需要对异常和中断承担哪些职责；分不清哪些由硬件完成哪些由软件完成。因为 CPU 设计涉及异常和中断，所以，本章中应把 CPU、内部异常、外部中断和输入/输出控制这四者的关系交代清楚。主教材对内部异常和外部中断的基本概念，以及异常处理过程，进行了较为详细的说明，并结合多周期数据通路及其反映指令执行过程的有限状态机，对 CPU 中涉及的异常和中断处理功能及部件进行了说明。因为是结合具体数据通路进行说明，学生阅读起来应该比较容易明白。

6.2 主要内容提要

1. CPU 的基本功能

CPU 总是周而复始地执行指令，并在执行指令过程中检测和处理内部异常事件和外部中断请求。在此过程中，要求 CPU 具有以下各种功能：①取指令并译码。从存储器取指令，对指令操作码译码，以控制指令进行相应的操作。②计算 PC 的值。自动计算 PC 的值来确定下条指令地址，以正确控制指令的执行顺序。③算术逻辑运算。计算操作数地址，或对操作数进行算术或逻辑运算。④取操作数或写结果。通过控制对存储器或 I/O 接口的访问来读取操作数或写结果。⑤异常或中断处理。检测有无异常事件或中断请求，必要时响应并调出相应的处理程序执行。⑥时序控制。通过生成时钟信号来控制上述每个操作的先后顺序和操作时间。

2. CPU 的基本结构

CPU 主要由数据通路(Datapath)和控制单元(Control Unit)组成。

数据通路中包含组合逻辑单元和存储信息的状态单元。组合逻辑单元用于对数据进行处理，如加法器、ALU、扩展器(0 扩展或符号扩展)、多路选择器，以及总线接口逻辑等；状态单元包括触发器、寄存器等，用于对指令执行的中间状态或最终结果进行保存。

控制单元也称为控制器，主要功能是对取出的指令进行译码，并与指令执行得到的条件码或当前机器的状态、时序信号等组合，生成对数据通路进行控制的控制信号。

3. CPU 中的寄存器

CPU 中存在大量寄存器，根据对用户程序的透明程度可以分成以下 3 类。

(1) 用户可见寄存器

指用户程序中的指令可直接访问或间接修改其值的寄存器。包括通用寄存器、地址寄存器和程序计数器 PC。通用寄存器可用来存放地址或数据；地址寄存器专门用来存放首地址或指针信息，如段寄存器、变址寄存器、基址寄存器、堆栈指针、帧指针等；程序计数器 PC 存放当前或下一条指令的地址。

(2) 用户部分可见寄存器

指用户程序中的指令只能读取部分信息的寄存器，如程序状态字寄存器 PSWR 或标志寄存器 FLAG，其内容由 CPU 根据指令执行结果自动设定，用户程序执行过程中可能会隐含读出其部分内容，以确定程序的执行顺序，但不能修改这些寄存器的内容。

(3) 用户不可见寄存器

指用户程序不能进行任何访问的寄存器，这些寄存器大多用于记录控制信息和状态信息，只能由 CPU 硬件或操作系统内核程序访问。例如，指令寄存器 IR 用来存放正在执行的指令，只能被硬件访问；存储器地址寄存器(MAR)和存储器数据寄存器(MDR)分别用来存放将要访问的存储单元的地址和数据，也由硬件直接访问；中断请求寄存器、进程控制块指针、页表基址寄存器等只能由内核程序访问，因此也都是用户不可见寄存器。

4. 指令执行过程

指令执行过程大致分为取指、译码、取数、运算、存结果、查中断等步骤。指令周期是指取出并执行一条指令的时间，它由若干机器周期或直接由若干时钟周期组成。早期的机器因为没有引入 cache，所以每个指令周期都要执行一次或多次总线操作，以访问主存取得指令或进行数据读写，因而将指令周期分成若干机器周期。每个机器周期对应某种 CPU 内部操作或某种总线事务，一个总线事务访问一次主存或 I/O 接口。一个总线事务包含送地址和读写命令、等待主存、读写数据等，故需要多个时钟周期才能完成，所以一个机器周期由多个时钟周期组成。现代计算机引入 cache 后，大多数情况下都不需要访问主存，而可以直接在 CPU 内的 cache 中读指令或访问数据，因此，每个指令周期直接由若干时钟周期组成。时钟是 CPU 中用于同步控制的信号，时钟周期是 CPU 中最小的时间单位。

5. 数据通路的实现

现代计算机都采用时钟信号进行定时，一旦时钟边沿信号到来，数据通路中的状态单元开始写入信息。不同指令其功能不同，所以，每条指令执行时数据在数据通路中所经过的路径及其路径上的部件都可能不同。不过，每条指令在取指令阶段所做的工作都一样。

根据是否将所有部件通过总线相连，可将数据通路分成总线式数据通路和非总线式数据通路；根据指令执行过程是否按流水线方式进行，可将数据通路分成非流水线数据通路和流水线数据通路。总线式数据通路无法实现指令流水线，所以，一定是非流水线数据通路。现代计算机都采用流水线数据通路。

总线式数据通路中，寄存器和 ALU 的输入、输出端之间都通过 CPU 内部总线交换数据，因为一个总线上某一时刻只能传送一个部件送出的信息，所以，总线式 CPU 中执行指

令时，每一步都只能串行进行，速度很慢。若所有部件连接到一个总线上，则是单总线数据通路；还可以将 ALU 的输入和输出端分别连到不同的两个总线上，构成双总线数据通路；还可以将 ALU 的两个输入端和一个输出端分别连接到 3 个总线上构成三总线数据通路。

非总线式数据通路可设计成单周期数据通路、多周期数据通路和流水线数据通路。以下用例子来说明单周期和多周期数据通路的实现。流水线数据通路在第 7 章介绍。

6. 数据通路设计举例

以下以 MIPS 指令为例，概要介绍单周期和多周期数据通路的设计原理和设计步骤。

(1) 确定实现目标。为方便起见，假定实现目标如表 6.1 所示的 11 条 MIPS 指令。

表 6.1　11 条目标指令及其功能描述

指　　令	功　　能	说　　明
add rd, rs, rt sub rd, rs, rt	R[rd]←R[rs]±R[rt]	从 rs、rt 中取数后相加/减，若溢出则异常处理，否则，结果送 rd
subu rd, rs, rt	R[rd]←R[rs]－R[rt]	从 rs、rt 中取数后相减，结果送 rd(不进行溢出判断)
slt rd, rs, rt	if (R[rs]<R[rt]) R[rd]←1　else R[rd]←0	从 rs、rt 中取数后按带符号整数判断两数大小。若小于则 rd 中置 1，否则，rd 中清“0”(不进行溢出判断)
sltu rd, rs, rt	if (R[rs]<R[rt]) R[rd]←1　else R[rd]←0	从 rs、rt 中取数后按无符号数判断两数大小。若小于则 rd 中置 1，否则，rd 中清“0”(不进行溢出判断)
ori rt, rs, imm16	R[rt]←R[rs] | ZeroExt(imm16)	从 rs 取数，将立即数 imm16 进行零扩展，然后两者按位“或”，结果送 rt
addiu rt, rs, imm16	R[rt]←R[rs]＋SignExt(imm16)	从 rs 取数，将立即数 imm16 进行符号扩展，然后两者相加，结果送 rt(不进行溢出判断)
lw rt, rs, imm16	Addr←R[rs]＋SignExt(imm16) R[rt]←M[Addr]	从 rs 取数，将立即数 imm16 进行符号扩展，然后两者相加，结果作为访存地址 从存储单元 Addr 中取数并送 rt
sw rt, rs, imm16	Addr←R[rs]＋SignExt(imm16) M[Addr]←R[rt]	从 rs 取数，将立即数 imm16 进行符号扩展，然后两者相加，结果作为访存地址 从寄存器 rt 取数并送存储单元 Addr 中
beq rs, rt, imm16	Cond←R[rs]－R[rt] if (Cond eq 0) PC←PC＋(SignExt(imm16)×4)	作减法以比较 rs 和 rt 中内容的大小 计算下条指令地址(根据比较结果，修改 PC) 转移目标地址采用相对寻址，基准地址为下条指令地址(即 PC＋4)，位移量为立即数 imm16 经符号扩展后的值的 4 倍
j target	PC<31:2>←PC<31:28> || target<25:0>	第一步无须进行 PC＋4 而直接计算目标地址，符号 || 表示“拼接”

表 6.1 给出了每条指令功能的 RTL 描述。每条指令的取指阶段功能都一样，都需要从 PC 所指的内存单元取指令，并使 PC 加 4。为避免重复说明，表中省略了对取指阶段功能的描述。

图 6.1 给出了 3 种 MIPS 指令格式，表 6.1 中前 5 条是 R-型指令，随后 5 条是 I-型指令，最后一条跳转指令“j target”是 J-型指令。



图 6.1 MIPS 指令格式

(2) 设计 ALU 电路。对目标指令中涉及的所有运算进行分析，确定用于这些运算的 ALU 及其控制电路如何设计。从表 6.1 可以看出，这 11 条指令涉及的运算包括带溢出检测的加法和减法、带符号整数大小判断、无符号数大小判断、相等判断以及各种逻辑运算等。图 6.2 给出了实现这些运算的 ALU 电路。

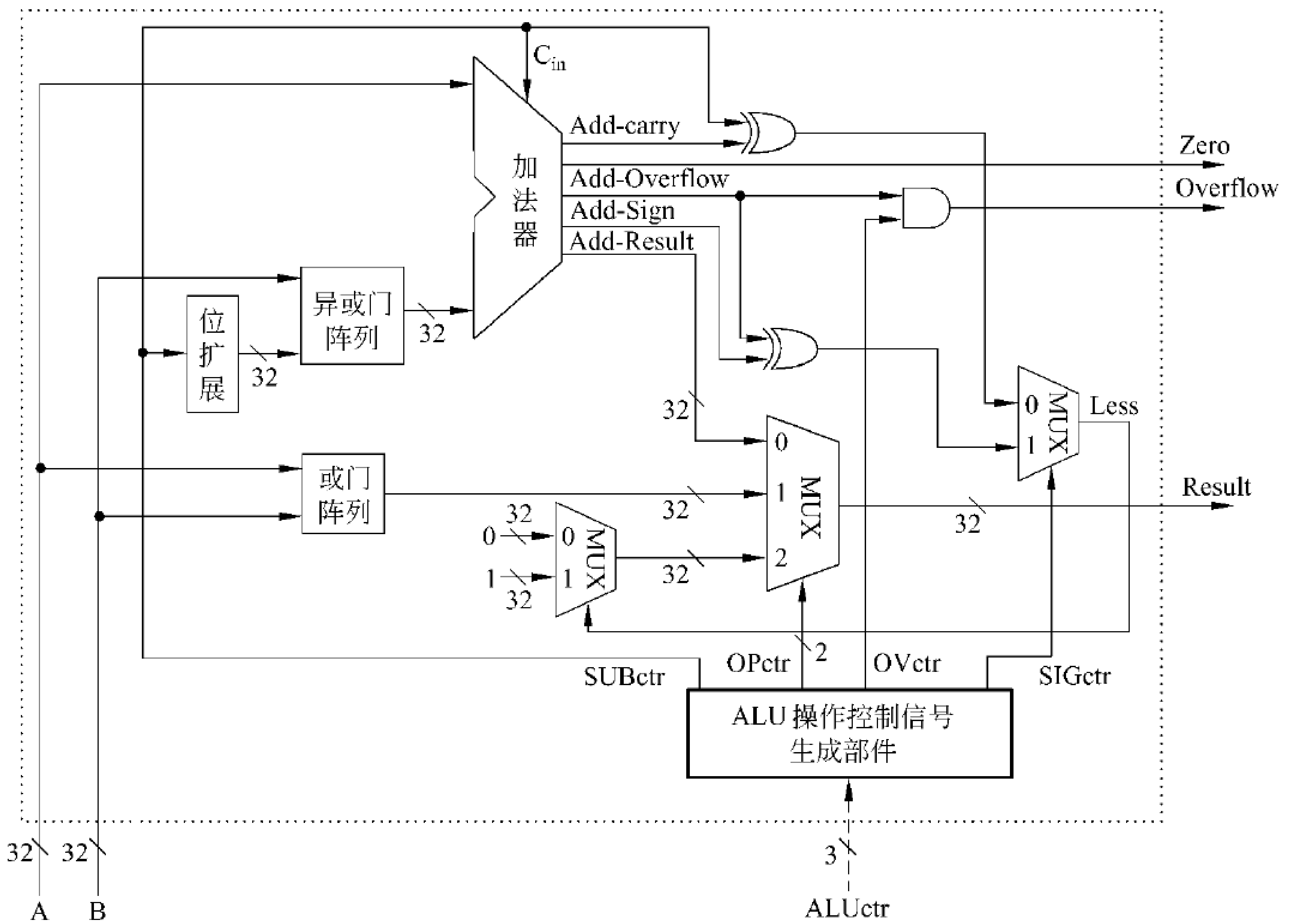


图 6.2 11 条目标指令的 ALU 实现

上述 ALU 中的核心部件是加法器，减法运算也在该加法器中实现，加法器有进位标志 Add-carry、零标志 Zero、溢出标志 Add-Overflow 和符号标志 Add-Sign。该 ALU 的输入为两个 32 位操作数 A 和 B，Result 作为 ALU 运算的结果被输出，同时，零标志 Zero 和溢出

标志 Overflow 也被作为 ALU 的结果标志信息输出。

为了实现对 ALU 操作的控制，需要有相应的操作控制信号。在操作控制端 ALUctr 的控制下，图 6.2 中的“ALU 操作控制信号生成部件”用来生成各种操作控制信号，以控制在 ALU 中执行“加法”、“减法”、“按位或”、“带符号整数比较小于置 1”和“无符号数比较小于置 1”等运算。

(3) 设计取指令部件。取指令操作是每条指令的公共操作，其功能是取指令并计算下一条指令地址。若是顺序执行，则下一条执行指令地址为 PC＋4；若是转移执行，则要根据当前指令是分支指令还是跳转指令分别计算转移目标地址。因为指令长度为 32 位，主存按字节编址，所以指令地址总是 4 的倍数，即最后两位总是 00，因此，PC 中只需存放前 30 位地址 PC＜31:2＞，在取指令时，指令地址＝PC＜31:2＞ ‖“00”。下条指令地址的计算方法如下。

顺序执行指令时：PC＜31:2＞←PC＜31:2＞＋1。

Branch 指令条件满足时：PC＜31:2＞←PC＜31:2＞＋1＋SignExt[imm16]。

Jump 指令跳转执行时：PC＜31:2＞←PC＜31:28＞ ‖ target＜25:0＞。

根据上述指令地址计算方法，图 6.3 给出了完整的取指令部件。

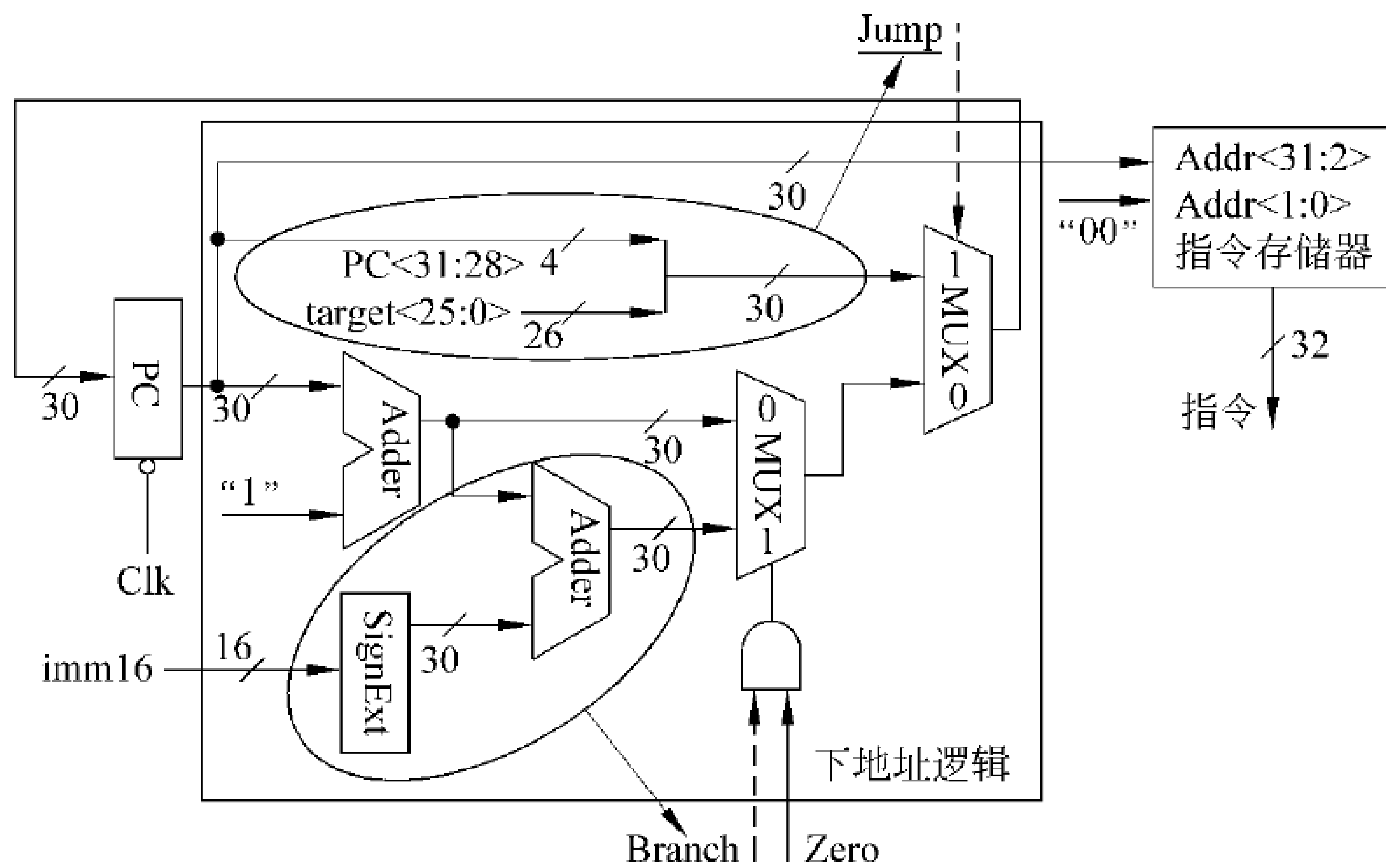


图 6.3 完整的取指令部件

取指令部件的输出是指令，输入有 3 个：标志 Zero 和控制信号 Branch、Jump。下地址逻辑中的立即数 imm16 和目标地址 target＜25:0＞都直接来自取出的指令。分支指令时，Branch＝1，Jump＝0；跳转指令时，Branch＝0，Jump＝1。

(4) 单周期数据通路设计。11 条指令中，lw 指令最复杂，执行 lw 指令过程中数据所经过的部件最多，路径最长，因此，以它为基准设计单周期数据通路。图 6.4 给出了能够执行 11 条目标指令的完整的单周期数据通路。

图 6.4 中带下划线的是控制信号，用于控制数据通路的执行，由专门的控制逻辑单元根据当前指令的译码结果生成控制信号。

(5) 时钟周期的确定。单周期处理器每条指令在一个时钟周期内完成，所以 CPI 为 1，时钟周期通常取最复杂指令所花的指令周期。在给出的 11 条指令中，最长的是 lw 指令周期。图 6.5 给出了 lw 指令执行定时过程，从图中可以看出，lw 指令周期所包含的时间为

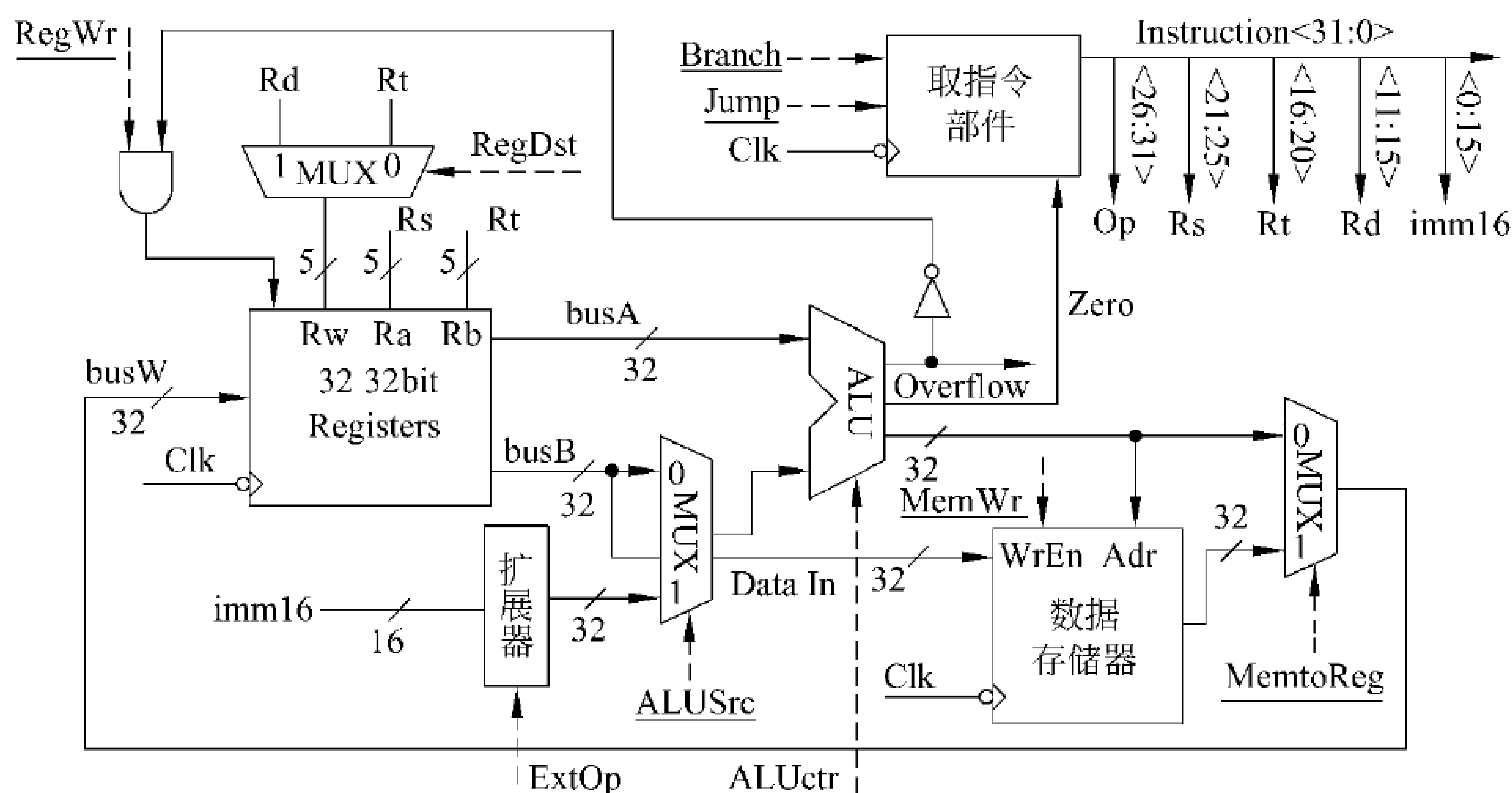


图 6.4　完整的单周期数据通路

PC 锁存延迟(Clk-to-Q)＋取指令时间＋寄存器取数时间＋ALU 延迟＋存储器取数时间＋寄存器建立时间＋时钟偏移。

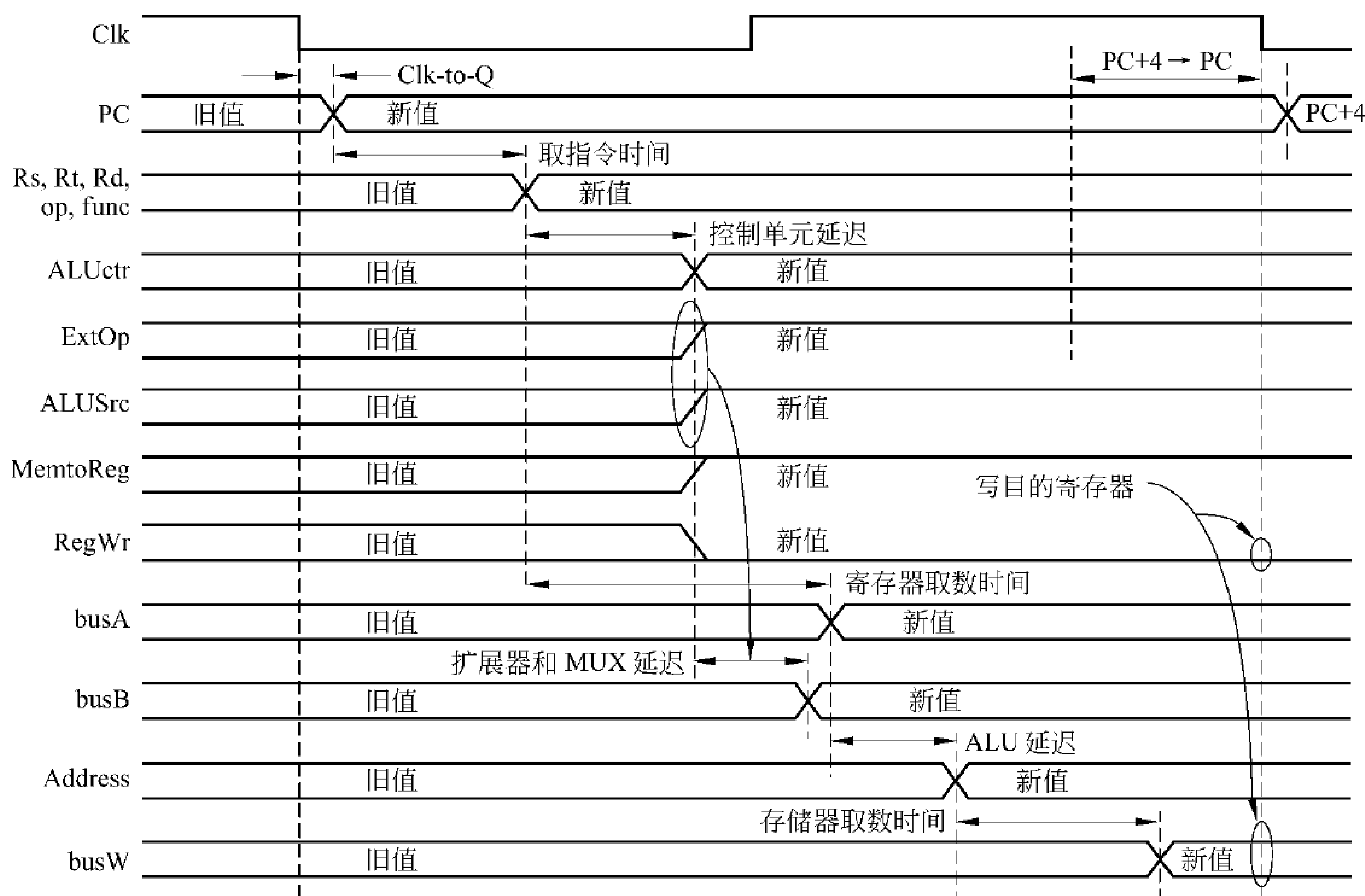


图 6.5　lw 指令执行定时

单周期处理器时钟周期远远大于许多指令实际所需执行时间，例如，R-型指令和立即数运算指令都不需要读内存；sw 指令不需要写寄存器；分支指令不需要访问内存和写寄存器；跳转指令不需要 ALU 运算和内存(寄存器)的读写。因而，单周期处理器的效率低下，性能极差，实际上，现在很少用单周期方式设计 CPU。介绍单周期数据通路，只是为了帮助

理解实际的多周期和流水线两种方式。

(6) 多周期数据通路设计。多周期处理器的基本思想为：把每条指令的执行分成多个大致相等的阶段，每个阶段在一个时钟周期内完成；各阶段内最多完成1次访存或1次寄存器读/写或1次ALU操作；各阶段的执行结果在下一个时钟到来时保存到相应存储单元或稳定地保持在组合电路中；时钟周期的宽度以最复杂阶段所花时间为准，通常取一次存储器读写的时间。图6.6给出了实现11条目标指令的多周期数据通路，其中带下划线的是控制信号。

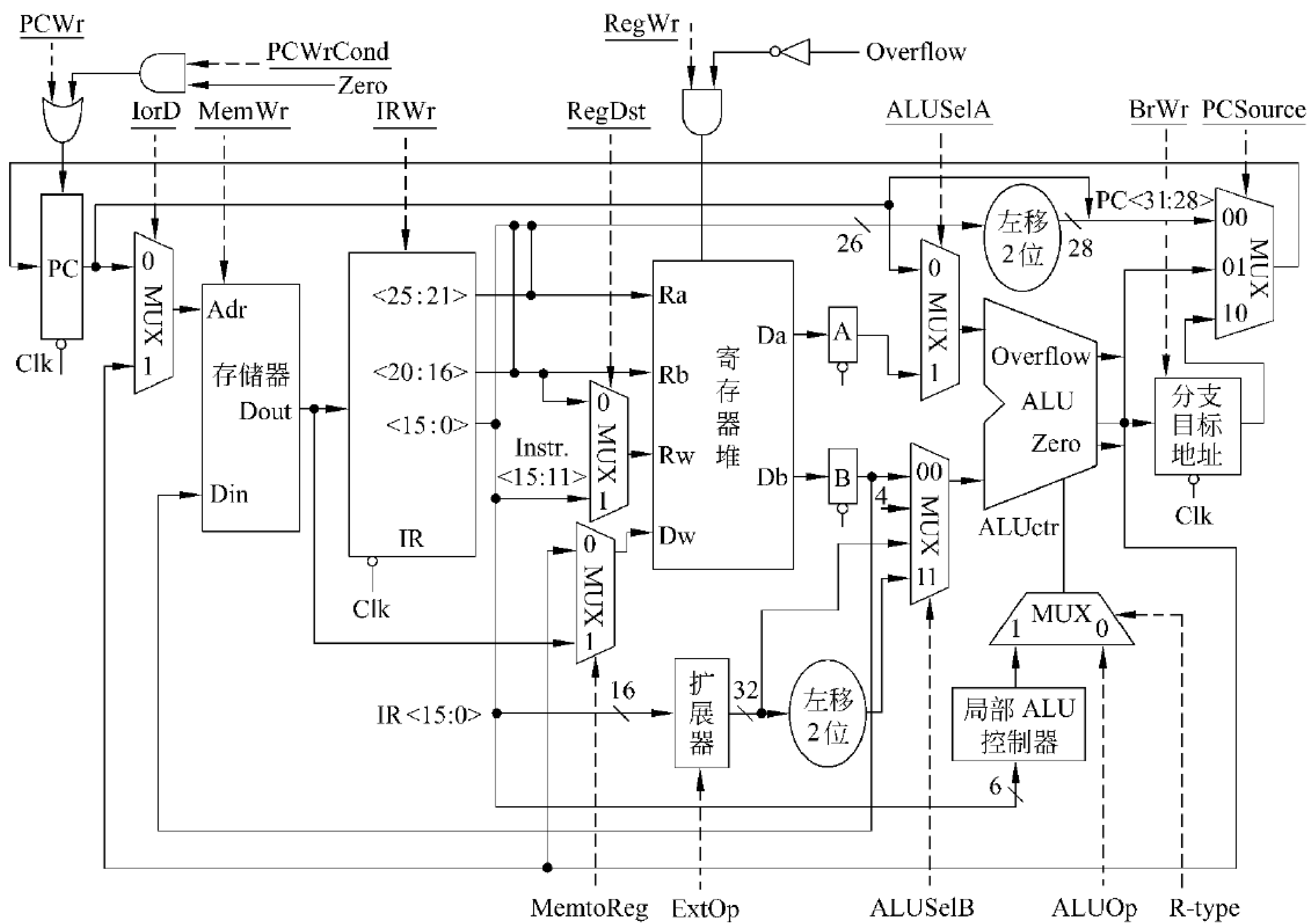


图6.6 带控制信号的多周期数据通路

(7) 分析每条指令的执行过程，得到指令执行状态转换图。每条指令在取指令周期(IFetch)和取数/译码周期(RFetch/ID)所进行的操作完全一样。除了取指令和译码/取数两个周期外，在11条目标指令中，R-型指令还需要一个执行周期(RExec)和一个结束回写周期(RFinish)；ori指令也还需要一个执行周期(oriExec)和一个结束回写周期(oriFinish)；分支指令和跳转指令都是只需要一个周期，分别是BrFinish和JumpFinish；lw和sw共用一个主存地址计算周期(MemAdr)，然后根据指令是lw还是sw，确定后面是写主存周期(swFinish)，还是取数周期(MemFetch)和写寄存器周期(lwFinish)。11条指令在图6.6所示的多周期数据通路中执行时的状态转换过程如图6.7所示。

图6.7反映了指令在多周期数据通路中执行时的状态转换过程，每个周期对应一个状态。每来一个时钟，进入下一个执行状态。从图6.7可以看出，R-型指令、I-型运算类指令和sw指令的CPI都是4；lw指令的CPI为5；分支指令和跳转指令的CPI为3。

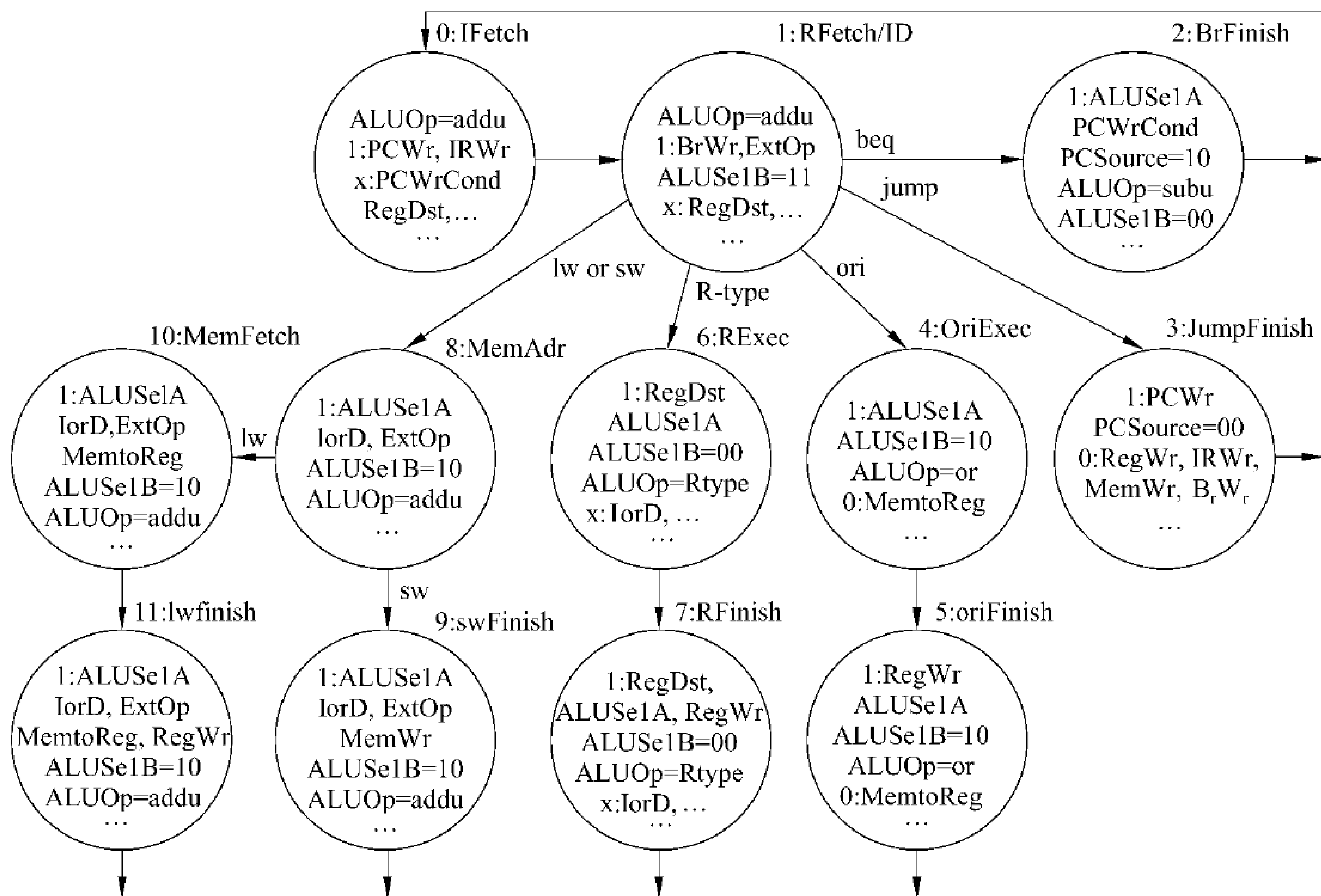


图 6.7　指令执行状态转换图

7. 控制逻辑单元的实现

根据不同的控制描述方式,可以有硬连线路控制器和微程序控制器两种实现方式。

硬连线路控制器的基本实现思路:将指令执行过程中每个时钟周期所包含的控制信号取值组合看成一个状态,每来一个时钟,控制信号会有一组新的取值,也就是一个新的状态,这样,所有指令的执行过程就可以用一个有限状态转换图来描述,如图 6.7 所示。实现时,用一个组合逻辑电路(一般为 PLA 电路)来生成控制信号,用一个状态寄存器实现状态之间的转换。

微程序控制器的基本实现思路:将指令执行过程中每个时钟周期所包含的控制信号取值组合看成是一个 0/1 序列,每个控制信号对应一个微命令,控制信号取不同的值,就发出不同的微命令。这样,若干微命令组合成一个微指令,每条指令所包含的动作就由若干条微指令来完成。指令执行时,先找到对应的第一条微指令,然后按照特定的顺序取出后续的微指令执行。每来一个时钟,执行一条微指令。实现时,每条指令对应的微指令序列(称为微程序)事先存放在一个只读存储器(称为控制存储器,简称控存)中,用一个 PLA 电路或 ROM 来生成每条指令对应的微程序的第一条微指令地址,用相应的微程序定序器来控制微指令执行流程。微程序定序器的实现有计数器法(增量法)和断定法(下址字段法)两种。

8. 内部异常和外部中断

在程序正常执行过程中,CPU 会遇到一些特殊情况而无法继续执行当前程序。这种中断程序正常执行的情况主要有两大类:内部异常和外部中断。

内部异常指 CPU 内部在执行某条指令时发生的程序异常或硬件异常,有故障、陷阱和终止 3 种类型,也被称为程序性中断或软中断。(1)故障是指某条正在执行的指令产生的异

常，如“溢出”、“除数为0”、“非法操作码”、“缺页”、“地址越界”、“访问越权”、“段不存在”、“堆栈溢出”等，有些故障修复后程序可以继续执行下去，有些故障不能修复，只能中止发生异常的程序的执行，若修复后程序能继续执行下去，则异常处理后通常回到发生故障的指令重新执行；(2)陷阱是预先安排的一种“异常”事件，例如，由断点设置、单步跟踪、系统调用、条件自陷等引起，通常异常处理后回到被中断处下一条指令执行；(3)终止指严重的硬件故障，一旦发生只能终止整个系统的运行，重启系统。

外部中断是指外设通过中断请求线向CPU提出的处理请求，它与指令的执行无关。它作为外设的一种I/O方式，能带来CPU和外设在一定程度上的并行性。有关外部中断的内容主要在第9章介绍。

6.3 基本术语解释

指令周期(Instruction Cycle)

CPU总是周而复始地取出指令并执行。因此，把取出一条指令并执行完所用的全部时间称为指令周期。一个指令周期中要完成多个步骤的操作，包括取指令、指令译码、计算操作数地址、取操作数、运算、送结果、中断检测等。

机器周期(Machine Cycle)

在指令周期中，最复杂的操作是访问存储器以获取指令或读/写数据，以及访问I/O。在没有片内cache的情况下，它们都需通过系统总线来和CPU之外的部件进行信息交换。因此，通常把CPU通过一次总线事务访问一次主存或I/O的时间称为机器周期。

一个指令周期包含了多个机器周期。不同机器的指令周期所包含的机器周期数不同。典型的机器周期有：取指令、主存读(间址周期是一种主存读机器周期)、主存写、I/O读、I/O写、中断响应等。一台计算机的机器周期类型是确定的。

现代计算机采用CPU片内cache来存放指令和数据，因此指令和数据的读取、数据的运算和传输都非常快，所以，一条指令的执行在一个或若干个时钟内就可以完成，不再将指令周期细分为若干机器周期。

同步系统(Synchronous System)

系统中所有的动作都有专门的时序信号来定时，最基本的时序信号就是时钟信号，同步系统通过时钟信号来规定何时发出什么动作。例如，在CPU内部，一个指令的执行要完成数据的读/写、传送或运算等。因此，指令的执行过程必须分解成若干步骤和相应的动作来完成，每一步动作都要有相应的控制信号进行控制，这些控制信号何时发出、作用时间多长，都要有相应的定时信号进行同步，这个定时信号就是时钟信号。

时序信号(Timing Signal)

同步系统中用于进行同步控制的定时信号。早期计算机的处理器设计时，采用机器周期-节拍-工作脉冲三级时序系统。现代计算机一般只用一个专门的时钟信号来进行定时。因此，现代计算机的时序信号就是时钟信号。

控制单元(Control Unit,CU)

也称为控制部件、控制逻辑或控制器。其作用是对指令进行译码，将译码结果和状态/标志信号和时序信号等进行组合，产生各种操作控制信号。这些控制信号被送到CPU内

部或通过总线送到主存或I/O模块。送到CPU内部的控制信号用于控制CPU内部数据通路的执行，送到主存或I/O模块的信号控制CPU和主存或CPU和I/O模块之间的信息交换。控制单元是整个CPU的指挥控制中心，通过规定各部件在何时做什么动作来控制数据的流动，以完成指令的执行。

执行部件(Execute Unit,EU)

也称为操作部件或功能部件，由控制部件CU发来的操作控制信号进行控制，以完成特定的功能。有两种类型的执行部件，一种是用组合逻辑电路实现的“操作元件”，用于进行数据运算、数据传送等，如ALU、总线、扩展器、多路选择器等；另一种是用时序逻辑电路实现的“状态元件”，用于进行数据存储，如触发器、寄存器、存储器等。

组合逻辑电路(Combinational Logic Circuit)

简称组合电路，用来构成“操作元件”。组合逻辑电路在逻辑功能上的特点是，任意时刻的输出仅仅取决于该时刻的输入，与电路原来的状态无关，因此，它没有存储功能。

时序逻辑电路(Sequential Logic Circuit)

简称时序电路，用来构成“状态元件”。时序逻辑电路在逻辑功能上的特点是，任意时刻的输出不仅取决于当时的输入信号，而且还取决于电路原来的状态，或者说，还与以前的输入有关。时序逻辑电路具有存储功能，能保存所存储的状态。

扩展器(Extension Unit)

将一个n位数扩展成2n位数的部件。一般有两种扩展方式：“零”扩展和“符号”扩展。

“零”扩展(0-Extend)

高n位用“0”填充。例如，在MIPS指令系统中，一个16位的逻辑数需先进行“0”扩展，扩展为32位数后，在32位ALU中进行逻辑运算。“0”扩展可以看成是对无符号数的扩展，扩展前后数值不变。

“符号”扩展(Sign Extend)

高n位用扩展前的n位数的符号位填充。例如，在MIPS指令系统中，Load/Store指令中的存储器地址偏移量需先进行“符号”扩展，将16位偏移量扩展为32位后，送到32位的ALU中，和基址进行加法运算。“符号”扩展可以看成是对有符号数扩展，扩展前后数值不变。

多路选择器(Multiplexor)

也称数据选择器或多路复用器。它根据控制线路的设置，选择多个输入信号中的一个进行输出。

定时方式(Clocking Methodology)

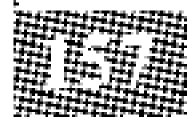
在时序电路中，定时方式规定了状态存储元件何时读出信号、何时写入信号。一般采用边沿触发的定时方式。

边沿触发(Edge-triggered)

规定存储元件中的状态只允许在时钟跳变边沿改变。时钟信号的跳变有正跳变和负跳变两种。在时钟的上升沿进行的跳变为正跳变；在时钟的下降沿进行的跳变为负跳变。

寄存器堆(Register File)

寄存器堆就是寄存器集合，所以也称为通用寄存器组(GRS)。其中的寄存器可以通过给定相应的寄存器编号来进行读写。在指令中用编号标识寄存器。执行指令时，指令中的



寄存器编号被送到一个地址译码器进行译码来选中某个寄存器进行写入，读出时寄存器编号作为控制信号来控制一个多路选择器，选择相应的寄存器读出。实质上它是CPU中暂时存放数据的地方，里面保存着那些等待处理的数据，或已经处理过的数据，CPU访问寄存器所用的时间要比访问内存的时间短。采用寄存器，可以减少CPU访问主存的次数，从而提高了CPU的工作速度。但因为受到芯片面积和集成度的限制，寄存器堆的容量不可能很大。

寄存器写信号(Register Write)

寄存器堆中的寄存器是由触发器构成的，而触发器的输出状态的变化只能发生在时钟边沿，因此寄存器写控制信号在时钟边沿有效。时钟边沿到来时，事先稳定在输入端的数据开始向寄存器写入，经过一段时间延迟(这个延迟时间称为Click-to-Q)才能稳定地写入寄存器，时钟边沿到来时，在寄存器输出端的数据还是上一个时钟周期内的老数据。

指令存储器(Instruction Memory)

专门存放指令的存储器，也称为代码存储器(Code Memory)。实际上，现代计算机中，CPU内的一级cache采用数据cache和代码cache分离的方式，因此指令存储器实际上是CPU中的代码cache。

数据存储器(Data Memory)

专门存放数据的存储器。实际上，现代计算机中，CPU内的一级cache采用数据cache和代码cache分离的方式，因此数据存储器实际上是处理器中的数据cache。

指令译码器(Instruction Decoder)

用来对指令的操作码进行译码的部件。在设计指令系统时，对每条功能不同的指令操作码进行了编码。因此，执行指令时，必须要有相应的译码线路对每个操作码进行译码解释。指令译码器的输入是指令操作码，输出结果用来和其他信号(如时序信号、机器状态信号、指令结果标志信号等)一起组合生成控制信号。不同的指令译码结果生成不同的控制信息，以规定执行部件做不同的动作。

控制信号(Control Signal)

也称为操作控制信号或微操作信号。控制单元对指令进行译码，将译码结果和状态/标志信号、时序信号等进行组合，产生各种操作控制信号。这些控制信号被送到CPU内部或通过总线送到主存或I/O模块。送到CPU内部的控制信号用于控制CPU内部数据通路的执行，送到主存或I/O模块的信号控制CPU和主存或CPU和I/O模块之间的信息交换。

时钟周期(Clock Cycle)

现代计算机的CPU采用时钟信号进行定时控制。若采用时钟边沿触发，则只有时钟的上升沿或下降沿到来此时，才能把一个新的值写到一个状态单元中。所以CPU的时钟周期应该为所有相邻状态单元之间的组合逻辑电路中最长的延时，以保证在一个时钟周期内所有的组合电路能完成必要的数据处理工作。

主频(CPU Clock Rate/Frequency)

CPU的主频就是CPU时钟周期的倒数。

分支条件满足(Branch Taken)

对于条件转移指令(分支指令Branch)，其执行结果总有两种可能性(两个分支或两条

执行路径)：一种是条件满足(称为 Branch taken),此时转到转移目的地址处继续执行；另一种是条件不满足(称为 Branch not taken),此时继续取下条指令执行。

转移目标地址(Branch Target Address)

转移指令(包括条件转移指令、无条件转移指令、转子指令等)中给出的目标地址称为转移目标地址。数据通路中必须要有相应的部件能计算转移目标地址,并根据情况选择送到PC 中作为下一条执行指令的地址。

硬布线控制器(Hardwared Control)

用组合逻辑方式进行设计和实现的控制器,也称为硬连线路控制器。在多周期数据通路中,一条指令的执行分多个阶段进行,每个阶段在一个时钟周期内完成,如果把每个阶段涉及的控制信号组合看成是一个状态,则一个阶段转移到下一个阶段,就可以看成是状态之间的转换,因此,所有指令的执行过程就可以用一个有限状态机来描述,控制器的功能就是实现这个有限状态机。因此,这种控制器设计方式也称为有限状态机方式,实现的控制器称为有限状态机控制器。它的优点是速度快,适合实现简单或规整的指令系统。但是,因为它是一个多输入/多输出的巨大的逻辑网络,所以,对于复杂的指令系统来说,其结构庞杂,实现困难,修改、维护不易,灵活性差。

微程序控制器(Microprogrammed Control)

采用微程序设计方式实现的控制器称为微程序控制器。微程序设计的基本思想是,仿照程序设计的方法编制每个机器指令对应的微程序,每个微程序由若干条微指令构成,每条微指令就是有限状态机中对应的一个状态,它是若干控制信号的一个组合,所以每个微指令包含若干微命令,这些微命令即控制信号。所有指令对应的微指令序列都放在一个只读存储器中。当执行到某条指令时,取出对应的各条微指令,译码产生对应的微命令(控制信号),送到机器相应的地方,控制其动作。这个只读存储器称为控制存储器(Control Storage,CS),简称控存。

微代码(Microcode)

通常把事先存放在控制存储器 CS 中的微程序代码(控存单元中的 0/1 序列)称为微代码或微码。

微指令(Microinstruction)

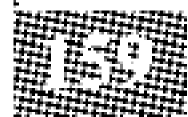
微指令和微代码的含义实际上是一样的,只是同一个概念从不同的角度来讲而已。控存中每个单元存放一条微指令,对应于有限状态机中的一个状态,每条微指令在一个时钟周期内完成。“微指令”与程序设计中“指令”的概念类似,也涉及格式和顺序控制等问题。

微程序(Microprogram)

类似于程序设计中“程序”的概念,程序用于实现某个特定的算法功能,而微程序用于实现机器指令;程序由若干指令构成,而微程序由若干条微指令构成;程序存放在内存中,而微程序存放在控制存储器中。

固件(Firmware)

一般把写入 EPROM 等只读存储器中的程序称为固件。最初把固化在只读存储器的微程序称为固件,表示用软件实现的硬部件,现在对固件通俗的理解就是在 ROM 中“固化的软件”。它是固化在集成电路内部的程序代码,负责控制和协调集成电路的功能。



中断过程(Interrupt Processing)

中断过程是一个正常执行的程序被打断的过程。指在程序正常执行过程中,CPU 遇到一些异常情况无法继续执行当前指令,或者,外部设备发生一些特殊事件请求 CPU 处理。此时,CPU 中止原来正在执行的程序,转到处理异常情况或特殊事件的处理程序去执行,执行后再返回到原被中止的程序继续执行。中断过程的起因主要有来自处理器外部的"中断"和来自处理器内部的"异常"两种。

中断(Interrupt)

引起中断过程的原因之一。在程序执行过程中,若外设完成任务或发生某些特殊事件(如打印机缺纸、定时采样计数时间到、键盘缓冲满等),会向 CPU 发中断请求,要求 CPU 对这些情况进行处理。处理完后,回到原被中断的断点处继续执行。这种情况也被称为 I/O 中断(I/O Interrupt)。特指由 CPU 外部的 I/O 设备向 CPU 发的中断请求。它与执行的指令无关,是异步发生的外部事件。因此,也称为"外部中断"。

异常(Exception)

引起中断过程的原因之一。在 CPU 执行某条指令时发生的一些特殊的非正常事件(如缺页、溢出、除数为 0、非法操作码等)都称为是一种异常。它是来自处理器内部的意外事件,和执行指令同步发生。也称为"内部异常"或"内中断"或"程序性中断",它又可以细分为 3 类:故障、自陷、终止。

故障(Fault)

在执行某条指令时,可能发生一些特殊的"异常事件",如缺页、溢出、除数为 0、非法操作码等,使当前指令无法继续执行。此时 CPU 只能中止原程序的执行,转到处理相应情况的程序去执行。有些故障处理完后,可再回到发生异常的指令继续执行,如缺页;有些故障无法解决,只好终止发生故障的进程。

自陷(Trap)

自陷是人为设定的事件,在程序中事先设定一条特殊的指令,通过执行这条特殊指令,自动终止正在执行的原程序,转到一个特定的内核管理程序去执行,执行完后,回到那条特殊指令后面的一条指令开始执行。这种情况称为自愿中断或自陷(Trap)。这些特殊的指令称为"访管指令"(访问管理程序)或"自陷指令"(自动掉入陷阱),如 80x86 中的指令"INT n"。

终止(Abort)

既不是外部设备发出的中断请求,也不是指令本身产生的异常情况或自愿中断,而是在执行指令过程中发生的硬件故障,如电源掉电、线路故障等。这类异常是随机发生的,对引起异常的指令的确切位置无法确定。出现这类严重错误时,程序无法继续执行,只好终止,由中断服务程序重新启动操作系统。

中断服务程序(Interrupt Handler)

也称为中断处理程序、异常处理程序等。当 CPU 发现中断或异常时,就会把当前正在执行的用户程序停下来,调出处理异常或中断的程序来执行,这个程序就是中断服务程序。中断服务程序属于操作系统内核部分,所以,发生中断或异常时,CPU 的状态要从用户态(即执行用户程序的状态,也称为目态)切换到管理态(即执行操作系统管理程序的状态,也称为管态)。

异常程序计数器 EPC(Exception PC)

MIPS 处理器中用于保存发生异常的指令或中断返回后所执行指令的地址的寄存器。它的位数和 PC 的位数一样。通常把这个被保存的地址称为断点,把这个地址送到 EPC 的过程称为保存断点。

原因寄存器(Cause Register)

用于记录异常或中断类型的状态寄存器。每一位的含义应有明确规定,例如:第一位为 1 则表示出现了"溢出"异常,第二位为 1 则表示出现了"非法指令",第三位为 1 则是"缺页",等等。在外部接口(如中断控制器)中也有类似的状态寄存器,用于记录中断请求的类型,一般称为中断请求寄存器。

中断允许触发器(Interrupt Enable Register)

在 CPU 中用来设置中断允许/中断禁止状态的触发器。关中断时,中断允许触发器被设置为 0,表示不允许响应中断。开中断时,中断允许触发器被设置为 1,表示允许响应中断。

关中断(Interrupt OFF)

将中断允许触发器设置为 0 的操作,表示不允许响应中断。

开中断(Interrupt ON)

将中断允许触发器设置为 1 的操作,表示允许响应中断。

中断向量(Interrupt Vector)

每个中断源都有对应的处理程序,称这个处理程序为中断服务程序,其入口地址称为中断向量。

中断响应(Interrupt Response)

是指从 CPU 发现有中断请求到取出中断服务程序的入口地址准备执行中断服务程序的过程。CPU 总是在一条指令执行完、取下条指令之前去查询有无中断请求。如果此时是开中断状态,并有未被屏蔽的中断请求发生,则 CPU 自动进入中断响应周期,执行一条隐指令,以完成关中断、保护断点、取中断向量 3 个操作。

向量中断(Vector Interrupt)

是指一种识别中断源的技术或方式。识别中断源的目的就是要找到中断源对应的中断服务程序的入口地址,即获得向量地址。采用向量中断进行中断源识别的做法是,采用某种硬件排队线路(如菊花链、并行判优等),对所有未被屏蔽的中断请求进行排队,选出优先级最高的中断源,然后对其编码,得到该中断源的编号,也称为中断类型号(可以转换为向量地址,有些书中就称其为向量地址),通过总线将其取到 CPU 中,转换成向量地址,从而取出中断服务程序入口地址,或跳转到中断服务程序。还有一种中断源识别方式是用程序(称为中断查询程序)进行识别的软件方法。

中断向量表(Interrupt Vector Table)

所有中断(包括异常)的中断服务程序入口地址构成一个表,称为中断向量表。有的机器把中断服务程序入口的跳转指令构成一张表,称为中断向量跳转表。

向量地址(Vector Address)

中断向量表或中断向量跳转表中每个表项所在的内存地址,称为向量地址。

中断类型号(Interrupt Number)

中断向量表或中断向量跳转表中每个表项所在的表项索引值,称为中断类型号。

6.4 常见问题解答

1. 从事 CPU 设计的人很少,为什么大家都要学习如何设计 CPU 呢?

答:首先,从智力方面来说,处理器设计是有趣而富有挑战性的,“现代微处理器可以称得上是人类创造出的最复杂的系统之一”,“处理器要完成复杂的任务,但又要求结构尽可能简单”。其次,理解处理器如何工作能够帮助理解整个计算机系统是如何工作的。再次,虽然没有很多人设计处理器,但会有很多人从事嵌入式系统的设计。现代许多机电产品中都有处理器芯片嵌入其中,嵌入式系统的设计者必须了解处理器是如何工作的,因为这些系统通常在较低的抽象层次上进行编程设计。最后,你将来的工作可能就是设计处理器。(引自[美] Randal E. Bryant & David O'Hallaron《Computer Systems A Programmer's Perspective》第 4 章)

现代计算机的处理器基本上都采用流水线方式执行指令,流水线方式的数据通路比较复杂,所以先从简单的单周期数据通路和多周期数据通路开始理解,这也就是本章的内容。有了这些基础,对流水线处理器理解起来就较容易了。深刻理解流水线方式处理器的工作原理对于如何设计高质量的程序、如何进行编译优化、如何设计高性能计算机系统等都是非常必要的。

2. 一条指令的执行过程包含哪些操作呢?

答:一条指令的执行过程包括取指令、指令译码、计算操作数地址、取操作数、运算、送结果。其中取指令和指令译码是每条指令都必须进行的操作。有些指令需要到存储单元取操作数,因此,需要在取数之前计算操作数所在的存储单元地址。取操作数和送结果这两个步骤,对于不同的指令,其取和送的地方可能不同,有些指令要求在寄存器取/送数,有些是在内存单元取/送数,还有些是对 I/O 端口取/送数。因此,一条指令的执行阶段(不包括取指令阶段),可能只有 CPU 参与,也可能要通过总线去访问主存,也可能要通过总线去访问 I/O 端口。

3. CPU 总是在执行指令吗? 会不会停下来什么都不做?

答:CPU 的功能就是不断地周而复始地执行指令,而每条指令又都有不同的步骤,每个步骤在一定的时间内完成。因此,CPU 总是在不停地执行指令。有时会说,CPU 停止或 CPU 正在等待,什么事情也不做,事实上,CPU 还是在执行指令,只不过可能处于以下几种情况之一:(1)CPU 正在执行某条指令的过程中,发生了诸如 cache 缺失这样需要访问内存或 I/O 端口的事件,此时,当前正在执行的指令无法继续执行到下一步,因此,CPU 就处于等待状态,直到主存或 I/O 完成读写操作;(2)CPU 正在不断地通过执行指令以查询外设是否就绪,在查询过程中,CPU 什么实质性的工作都没有做;(3)CPU 正在执行一连串的空指令(NOP),什么实质性工作都没有做;等等。综上所述,CPU 不可能不在执行指令,只是指令的执行过程被阻塞了一段时间或执行了没有产生结果的指令。

4. CPU 除了执行指令以外,还做什么事情?

答:CPU 的工作过程就是周而复始地执行指令,计算机各部分所进行的工作都是由

CPU根据指令的要求来启动的。为了使CPU和外部设备能够很好地协调工作，尽量使CPU不等待，甚至不参与外部设备的输入和输出过程，CPU对外设的输入/输出控制采用了程序中断方式和DMA方式。这两种方式下，外部设备需要向CPU提出中断请求或DMA请求，因此，在执行指令过程中，CPU还要通过定时采样相应的引脚来查询有没有中断请求或DMA请求。如果有中断请求，CPU要进行一系列操作来完成从正在执行的用户程序到中断处理程序的切换；如果有DMA请求，还要给出响应DMA请求的一些控制信号。另外，CPU在一条指令的执行过程中，还可能发生一些异常事件，因此，也需要CPU能通过相应的动作，转换到异常事件处理程序去执行。

5. 对于CPU中的所有寄存器，用户都能访问吗？

答：不是的。CPU中的寄存器分为用户可访问寄存器和用户不可见寄存器。一般把用户可访问寄存器称为通用寄存器(GPR)。这些寄存器都有一个编号，在指令中用编号标识寄存器。因此执行指令时，指令中的寄存器编号要送到一个地址译码器进行译码，然后才能选中某个寄存器进行读写。通用寄存器可以用来存放操作数或运算结果，或作为地址指针、变址寄存器、基址寄存器等。

CPU中有一些寄存器是用户不可见的，没有编号，不能通过程序直接访问。如指令寄存器IR、程序状态字寄存器PSWR、存储器地址寄存器MAR、存储器数据寄存器MDR等。

对于程序计数器PC，它虽然是专用寄存器，没有编号，不能在指令中被明确指定，但用户可以通过转移类指令来修改其值，以改变程序执行的顺序。因此某种程度上PC属于用户可见寄存器。

6. CPU执行指令的过程中，其他部件在做什么？

答：计算机的工作过程就是连续执行指令的过程，整个计算机各个部分的动作都是由CPU中的控制部件CU通过对指令译码送出的控制信号来控制的。其他部件不知道自己该做什么，该完成什么动作，只有CPU通过对指令译码才知道。如果指令中包含对存储器或I/O端口的访问，则必须由CPU通过总线，把要访问的地址和操作命令(读/写)等信息送到存储器或I/O接口来启动相应的读或写操作。例如，若不采用cache，则每次指令执行前，都要通过向总线发出主存地址、主存读命令等来控制存储器取指令；若当前执行的是寄存器定点加法指令，则CU控制定点运算器进行动作；若是I/O指令，则CU会通过总线发出I/O端口地址、I/O读或写命令等来控制对某个I/O接口中的寄存器进行读写操作。所以说，CPU在执行指令时，其他部件也可能在执行同样的指令，只不过它们各司其职：CU负责解释指令和发出命令(控制信号)，而各个执行部件负责按命令具体完成自己的职责(如读写数据、传送信息等)。

7. 怎样保证CPU能按程序规定的顺序执行指令呢？

答：计算机的工作过程就是连续执行指令的过程，指令在主存中连续存放。一般情况下，指令被顺序执行，只有遇到转移指令(如无条件转移、条件分支、调用和返回等指令)才不按指令存放的顺序执行。当执行到非转移指令时，CPU中的指令译码器通过对指令译码，知道正在执行的是一种顺序执行的指令，所以就直接通过对PC加“1”(这里的“1”是指一条指令的长度)来使PC指向下一条顺序执行的指令；当执行到转移指令时，指令译码器知道正在执行的是一种转移指令，因而，控制运算器进行相应的地址运算，把运算得到的转移目标地址送到PC中，使得执行的下一条指令为转移到的目标指令。

由此可以看出，指令在主存中的存放顺序是静态的，而指令的执行顺序是动态的。CPU 能根据指令执行的结果动态改变程序的执行流程。

8. 在定长指令字格式的处理器中，如何读取指令？

答：定长指令字格式是一种规整型指令集，所有指令都有相同的长度，所以，指令的读取非常简单。每次都可以按照确定的字节个数从指令存储器读出指令。

9. 在定长指令字格式的处理器中，下一条指令地址如何计算？

答：定长指令字格式是一种规整型指令集，所有指令都有相同的长度。现代计算机大多以字节编址，因此，在计算下条指令的地址时，只要将 PC 中的当前指令地址加上指令的字节数就行了。如 MIPS 处理器的指令字都是 32 位，所以，每条 MIPS 指令占了 4 个内存单元。只要将 PC 的值加 4 就可以得到下条指令的地址。

10. 在变长指令字格式的处理器中，如何读取指令？

答：变长指令字格式是一种不规整型指令集，指令有长有短，每条指令所含的字节数不同。因此，在取当前指令时，可以每次按最长的指令长度来取。例如，如果最长指令长度为 6，则每次取 6 个字节，然后根据指令中特定位的规定，对指令中的各字段进行划分，确定指令包含的操作码字段、寄存器编码字段、立即数字段、直接地址字段或转移目标地址字段等。因为总是按最长指令字读取，所以每条指令总是包含在读出的字节中。

11. 在变长指令字格式的处理器中，下一条指令地址如何计算？

答：变长指令字格式是一种不规整型指令集，指令有长有短，每条指令所含的字节数不同。因此，在计算下条指令的地址时，应将当前指令地址（PC 的内容）加上当前指令的字节数。例如：80x86 处理器的指令字是变长的，每条指令从一个字节到多个字节不等。因此，下条指令地址通过一个专门的 PC 增量器来进行计算，这个 PC 增量器能根据指令中相关字段的值，确定 PC 应该加几。

12. 从处理器设计的角度，你认为定长指令字好还是变长指令字好？

答：从处理器设计的角度来看，定长指令字格式比变长指令字格式要好。特别是在取指令操作和计算下条指令地址方面，定长指令字可以大大简化处理器中取指令部件的设计。

13. CPU 的主频越高，运算速度就越快吗？

答：CPU 中的执行部件（定点运算部件、浮点运算部件）的每一步动作都要有相应的控制信号进行控制，这些控制信号何时发出、作用时间多长，都要有相应的时钟定时信号进行同步，CPU 的主频就是同步时钟信号的频率。

从直观上看，主频越高，每一步的动作就越快，CPU 的运算速度也就越快。例如，若两台具有相同 ISA 的机器的 CPI 都为 2，则主频为 500MHz 的机器在一秒钟内执行 2.5 亿条指令；而主频为 1GHz 的机器在一秒钟内执行 5 亿条指令。显然主频越高指令执行速度越快。

主频是反映 CPU 性能的重要指标，但它只反映一个侧面，而不是绝对的速度指标。对于具有不同 ISA 和不同 CPI 的两台计算机，不能简单地根据主频来衡量运算速度。例如，如果将一条指令所包含的操作步骤分得很多，使每一步操作所用时间很短，那么，定时用的时钟周期就很短，因而主频就高，但是，此时 CPI 也变大了，因而执行一条指令的时间并没有缩短。

14. CPU 中的控制器的功能是什么?

答: CPU 中的控制器主要用于产生执行各条指令所需要的控制信号。有两大类控制信号: CPU 内部控制信号和发送到系统总线上的控制信号。

15. 数据通路的功能是什么?

答: CPU 的基本功能就是执行指令,指令的执行过程就是数据在数据通路中流动的过程。数据在流动过程中,要经过一些执行部件进行相应的处理,处理后的数据要送到存储部件保存。简而言之,数据通路的功能就是通过对数据进行处理、存储和传输来完成指令的执行。

16. 数据通路是如何进行数据处理、数据传送、数据存储的?

答: 数据通路中的功能部件包括两种不同的逻辑单元: 进行数据处理的组合逻辑单元(操作元件)和进行数据存储的时序逻辑单元(状态单元)。组合逻辑单元的输出只取决于当前的输入,也就是说输入一旦改变,在一定的线路延迟后,输出就跟着变化,因而它只能完成一定的数据处理功能,不能存储数据,只有将处理后的新数据送到一个状态单元才能保存下来。所以,所有的数据处理单元都必须将处理后的输出结果写到状态单元中,而在处理前又必须从状态单元接收输入。因此,数据流动的起点是状态单元,经过一些组合逻辑部件,最终又流回状态单元保存。功能部件之间的输入/输出信息通过连线进行传送。

17. 数据通路中流动的信息有哪些?

答: 指令的执行过程就是数据通路中信息的流动过程。因此要理解数据通路中流动的信息类型,必须先考察指令的执行过程。因为每条指令的功能不同,所以其执行过程也不一样。但总体来说,指令执行过程中涉及的基本操作包括取指令并送指令寄存器、计算下一条指令地址、下条指令地址送 PC、取寄存器数据到 ALU 输入端、指令中的立即数送扩展器或 ALU 输入端、ALU 中进行运算(包括计算内存单元地址)、读内存数据到寄存器、寄存器数据写到内存、ALU 输出写到寄存器等。因此,在数据通路中流动的信息有 PC 的值、指令、指令中的立即数、指令中的寄存器编号、寄存器中的操作数、ALU 运算的结果、内存单元中的操作数等。

18. 控制信号是如何控制数据的流动的?

答: 指令的执行过程就是数据通路中信息的流动过程。数据通路中信息的流动受控制信号的控制。当前指令被取到指令寄存器 IR 后,指令的操作码部分被送到指令译码器进行译码,指令译码输出的信号和其他信号一起组合后生成控制信号。所以不同的指令得到不同的控制信号,以规定数据通路完成不同的信息流动过程。在数据通路中,各个功能部件中都有一些控制点,这些控制点接受不同的控制信号,就使得功能部件完成不同的操作。例如: ALU 的操作控制点接受“Add”、“Sub”、“And”、“Or”等不同的操作信号,控制 ALU 完成加、减、与、或等不同的操作。Load/Store 指令译码后把“Add”信号送到 ALU 控制点,控制 ALU 进行加法运算来计算内存单元的地址。再例如: 有些状态单元的写入操作由一个“写控制信号”来控制,如果某条指令不需要把信息写入寄存器或内存单元,那么,这条指令对应的译码信号就使这个“写控制信号”无效,从而控制不写入任何信息。

19. 如何保证一条指令执行过程中的操作按序执行?

答: 对于每条指令来说,其执行过程应该是有序的。例如: 对于运算类指令,其操作数一定要先取出来才能进行运算,运算结果一定是在最后才能写到目的寄存器中。对于

Load/Store型指令一定是先取出源寄存器的值，并先对立即数进行符号扩展，然后才能把它们送到ALU相加，计算出内存单元的地址，随后再访问内存单元取数或存数。因而必须有一个机制来控制一条指令的有序执行，那么，如何控制呢？主要是通过将指令执行的每一步按序送到相应的组合逻辑处理部件和存储信息的状态元件。在每个时钟周期中，组合逻辑部件在相应的控制信号的控制下进行数据处理或数据传送，而在时钟有效信号到达时状态单元保存中间结果。这样，经过若干个时钟周期，一条指令就在数据通路中执行完成了。

20. 多路选择器的作用与工作原理是什么？

答：多路选择器也称数据选择器或多路复用器。它的作用就是在多个输入中选择其中的一个作为输出。因此，它有多个输入端、一个输出端和一个控制端。控制端的控制信号位数由输入端的个数确定，2选1时，控制信号有1位，根据控制信号为0还是1，决定将输入端1还是2作为输出。3选1或4选1时，控制信号有2位，根据控制信号为00、01、10还是11，决定将输入端1、2、3或4作为输出。

21. 加法器(Adder)和ALU的差别是什么？

答：加法器只能实现两个输入的相加运算，而ALU可以实现多种算术逻辑运算。可以用门电路直接实现加法器，也可以通过对ALU的操作控制端固定设置为"加"操作来实现加法器。在数据通路中有些地方只需做加法运算，如地址计算时，这时就不需要用ALU，只要用一个加法器即可。

22. 指令存储器和数据存储器的差别是什么？

答：指令存储器和数据存储器的功能不一样，指令存储器专门用来存放指令，而数据存储器专门存放数据。所以，从指令执行过程来看，指令存储器只在取每条指令时执行读操作，每个指令周期都一样，而数据存储器只有在执行访存指令时才被访问，而且不仅可能有读操作也可能有写操作。指令存储器的读地址由PC提供，而数据存储器的读地址和写地址都由ALU的输出端提供，因为数据的地址需要通过基址和偏移量相加得到。数据存储器的写操作需要一个写控制信号(写使能信号Write Enable)进行控制。在现代计算机中，CPU内的一级cache采用数据cache和指令cache分离的方式，所以指令存储器实际上是处理器中的指令cache，数据存储器实际上是处理器中的数据cache。从这点来看，指令存储器和数据存储器的基本实现应该是一样的，都是遵循cache设计的一套方法来实现的。

23. 如何确定CPU的时钟周期？

答：在一个边沿触发的同步数字系统中，一个状态量的改变总是在时钟的上升沿或下降沿发生，称上升沿或下降沿的到来为时钟有效信号到达。一个状态量的改变必须满足以下3个条件：(1)新写入的数据已经生成并稳定在状态单元的输入端；(2)写使能信号有效；(3)时钟有效信号到达。在前面两个条件满足的情况下，一旦时钟有效信号到达，则输入数据开始写入状态单元，经过一定的延迟后，状态单元的输出变为新输入的数据。所以要能使电路正确工作，必须使新写入的数据在时钟有效信号到达前稳定在输入端，即时钟周期必须足够长，使得在状态单元之间进行数据处理的组合逻辑电路有足够的时间来得到新的数据。所以，应该以所有相邻状态单元之间的组合逻辑电路中最长的延时为基准来确定时钟周期，以保证在一个时钟周期内所有的组合电路能完成必要的数据处理工作，在下一个时钟到来后，数据能存储在状态单元中。

24. 如何确定单周期数据通路的时钟周期？

答：单周期数据通路指每条指令的执行在一个时钟周期内完成，即 CPI＝1。因此，在单周期数据通路中，每当一个时钟有效边沿到来时，开始一条指令的执行，所有指令都要求在一个时钟周期内完成，下一个时钟有效边沿到来时，上一个时钟周期完成的指令的结果被写到目的寄存器或存储器中，同时上一个时钟周期内计算出指令的地址被输入 PC，一段时间(Click-to-Q)延迟后，PC 的值被送到指令存储器，开始取指令并执行。所以，单周期数据通路的时钟周期应该以最复杂的指令所需要的执行时间为准来确定，以保证所有指令都能在一个时钟周期内完成。因为单周期数据通路中指令的执行结果总是在下一个时钟到来时才被写到状态单元，所以，整个指令执行过程都是在组合逻辑电路中进行的，没有中间结果需要保存。以 MIPS 中复杂的 lw 指令为例，一个时钟的长度应该包括 PC 锁存时间(PC's Clock-to-Q)、取指时间(Instruction Memory Access Time)、寄存器堆存取时间(Register File Access Time)、ALU 延时(ALU Delay)、数据存取时间(Data Memory Access Time)、寄存器建立时间(Register File Setup Time)和时钟偏移(Clock Skew)。

25. 单周期数据通路中，控制信号何时发出？

答：单周期数据通路中，每条指令在一个周期内完成，所有控制信号同时产生，在指令取出后，指令操作码字段及相关的控制字段送到控制逻辑，由控制逻辑统一得到各个控制信号，每个控制信号被送到相应的控制点，一直作用在数据通路中，直到下一条指令执行过程中产生新的控制信号。例如：假定作用在 ALU 上的控制信号 ALUCtrl 为"001"时，ALU 做加法；为"010"时，做减法；为"011"时做"与"操作；……，那么，执行加法指令"Add"时，操作信号 ALUCtrl 一直以"001"作用在 ALU 的操作控制端上，若下一条为减法指令"Sub"，则该指令被取出译码后从控制逻辑送出的 ALUCtrl 信号的取值就变为"010"，在执行减法过程中，操作信号 ALUCtrl 一直以"010"作用在 ALU 的操作控制端上。

26. 如何确定多周期数据通路的时钟周期？

答：多周期数据通路通过把指令的执行分成多个阶段来实现，即 CPI≥1。规定每个阶段在一个时钟周期内完成，时钟周期以最复杂的阶段所花时间为准，阶段的划分原则是：将一条指令的执行过程尽量分成大致相等的若干阶段。这样，可以使得时钟周期尽量短。

在多周期数据通路中，一条指令的执行由多个时钟周期来控制。不同的指令所含的时钟周期数不同。复杂指令所含的时钟数多于简单指令的时钟数。

27. 多周期数据通路中，控制信号何时发出？

答：多周期数据通路中，每条指令的执行分成若干个阶段完成，每来一个时钟，就进入到下一个阶段。因为在数据通路中每个不同的阶段将完成不同的功能，所以控制信号在每个时钟到来时改变其值。因此，和单周期数据通路不同，多周期数据通路中，同一个控制信号在一个指令周期中可能多次改变其值。

28. 为什么很少有机器采用单周期数据通路？

答：因为单周期数据通路以最复杂指令所需时间为标准来设计时钟周期，每条指令的执行时间都一样，是极大的浪费。

29. 单周期处理器的基本设计步骤是什么？

答：处理器的设计是一个相当复杂的过程，设计者必须在集成电路技术、指令集体系结

构、计算机性能评价等方面具有丰富的知识和相当的经验。但不管有多复杂，其基本步骤可以归纳为以下几步：(1)分析每条指令的功能，并用某种形式化方法(如 RTL-Register Transfer Language)来表示。(2)根据指令的功能确定所需要的元件，并考虑如何将它们互连。各部件互连后的电路就是数据通路。(3)确定每个元件所需要的控制信号的取值。(4)汇总所有指令所涉及的控制信号，生成一张反映指令与控制信号之间关系的表。(5)根据关系表得到每个控制信号的逻辑表达式，据此设计控制器电路。

30. 控制器有哪两种实现方式？各有何优缺点？

答：有两种实现方式：(1)用组合逻辑设计方式实现硬连线路控制器。(2)用微程序设计方式实现微程序控制器。硬连线路控制器的优点是速度快，适合实现简单或规整的指令系统。但是，它是一个多输入/多输出的巨大的逻辑网络。对于复杂的指令系统来说，其结构庞杂，实现困难，修改、维护不易，灵活性差。微程序控制器的特点是具有规整性、可维性和灵活性，但速度慢。为了结合两种方式的优点，很多处理器采用两者结合的方式来实现。例如：Intel 80x86 系统中综合使用了硬布线来处理简单指令，用微程序方式(微码)实现复杂指令。

31. 如何用组合逻辑设计方式实现硬连线路控制器？

答：用组合逻辑设计方式实现硬连线路控制器，也称为基于有限状态机方式。其基本思想是将所有指令执行的每个阶段(一个时钟周期内)所包含的控制信号的取值作为一个状态，每来一个时钟，则进入下一个阶段对应的状态，每个状态对应一组控制信号。

这样，每条指令的执行过程就由有限状态机的每个状态中包含的控制信号来控制。因此，控制器的设计也就是考虑怎样使得控制器每来一个时钟就从当前状态进入到下一状态，状态的改变又使得控制信号的值发生改变。有限状态机控制器通常由一个组合逻辑电路模块和一个状态寄存器组成。状态寄存器如何变化是由当前状态和当前指令决定的，而组合逻辑控制模块可以由一个 ROM 或一个 PLA 实现。

32. 微程序控制器设计的基本思想是什么？

答：仿照程序设计的方法编制每个机器指令对应的微程序，每个微程序由若干条微指令构成，各微指令包含若干条微命令。所有指令对应的微程序放在只读存储器中。当执行到某条指令时，取出对应的微程序中的各条微指令，并对微指令译码产生对应的微命令(控制信号)，送到机器相应的地方，控制其动作。

33. 有哪几种微指令格式设计风格？

答：微指令格式设计有两种风格：一种是水平型微指令，面向内部控制逻辑的描述，包括不译法(直接控制法)、字段直接编码(译)法、字段间接编码(译)法。其基本思想是把能同时执行的微命令尽量多地安排在一条微指令中。通常把能同时执行的微命令称为相容微命令。这种微指令格式的微程序短，微命令并行性高，适合较高速度的应用场合。但缺点是微指令长，编码空间利用率较低，并且编制较为困难。另一种是垂直型微指令，面向算法描述，也称为最小(或最短、垂直)编码(译)法。其基本思想是借鉴指令编码的思想，使得一条微指令只包含一两个微命令。这种风格的微指令短，编码效率高，格式与机器指令类似，故编制容易。其缺点是微程序长，一条微指令只包含一两个微命令，无并行，因而速度慢。

34. 如何找到指令对应的第一条微指令？

答：控存中存放着所有指令对应的微程序，因而控存中有成百上千条微指令。每一条指

令执行时，必须找到这条指令对应微程序所包含的微指令序列中的第一条微指令，然后按一定的顺序执行这个微指令序列，每个时钟执行一条微指令。那么，如何找到对应的第一条微指令呢？通常一条指令的执行总是从取指令开始，在取出指令之后进行指令译码，可以用 ROM 或 PLA 来实现一个调度表，其入口为指令译码输入，而输出为每条指令对应的微指令序列的首条微指令在控存中的地址。根据这个地址到控存中取出第一条微指令执行即可。

35. 如何控制微指令的执行顺序？

答：在找到指令对应的首条微指令后，就可以按照一定的顺序来执行指令对应的微指令序列。那么，如何控制处理器按照正确的顺序执行微指令序列呢？这就是微指令定序器的实现问题。可以有两种方式来确定下条微指令地址：计数器法（增量法）和下址字段法（断定法）。

计数器法的基本思想：借鉴指令定序器的实现思路，用一个专门的计数器（通常称为微程序计数器 μPC，对应于程序计数器 PC）来指定下一条需执行的微指令地址。在微指令序列的执行过程中，大部分情况下每条微指令的后续微指令都是确定的，这样，只要在编制微程序时把后续微指令存放在控存的后续相邻单元中，就可以按照计数器指定的顺序执行，每执行完一条微指令，计数器 μPC 就顺序增量一次；对于少数几个需要根据不同的指令操作码或操作数的不同寻址方式进行选择的情况，可以用一个调度表或在微程序中插入转移微指令（分支微指令）来实现。

下址字段法的基本思想：在每条微指令中，增加一个字段，用于指出下一条要执行的微指令的地址，在遇到有多个后续微指令（多分支）的情况时，采用一个调度表或相应的地址修改逻辑来实现多分支。

36. 中断（Interrupt）和异常（Exception）的区别是什么？

答：有关中断和异常，不同教科书或体系结构有不同的含义。有些作者或体系结构并不区分它们，把它们统称为中断，如 Intel 80x86 系统用中断指代所有的意外事件。而 PowerPC 用异常来指代意外事件，用中断表示指令执行时控制流的改变。在 MIPS 系统和一些经典的国外教科书中，“异常”和“中断”的概念是有区别的。根据事件来自处理器内部还是外部来区分，把执行指令过程中由指令本身引起的来自处理器内部的异常事件称为“异常”，把来自处理器外部的由外部设备通过“中断请求”信号向 CPU 申请的事件称为“中断”。

37. 除了有外设向 CPU 发中断请求要求中断 CPU 外，还有哪些情况会中断 CPU 正在运行的程序，转到其他相应的处理过程去执行？

答：以下 4 种情况发生时，会中断 CPU 正在运行的程序，转到其他相应的处理过程去执行。

(1) 在程序执行过程中，若外设完成任务或发生某些特殊事件（如打印机缺纸、定时采样计数时间到、键盘缓冲满等），会向 CPU 发中断请求，要求 CPU 对这些情况进行处理。处理完后，回到原被中断程序的断点处继续执行。这种情况称为 I/O 中断或外部中断，特指由 CPU 外部的设备向 CPU 发的中断请求。

(2) 在执行某条指令时，可能发生一些特殊的“异常事件”，如缺页、溢出、除数为 0、非法操作码等，使当前指令无法继续执行。此时也要求 CPU 中止原程序的执行，转到处理相应情况的程序去执行，处理完后，再回到发生异常的指令继续执行。这种情况，称为失效或故

障(Fault),是由正在执行的指令产生的。

(3) 还有一类是人为设定的事件,在程序中事先设定一条特殊的指令,通过执行这条特殊指令,自动中止正在执行的原程序,转到一个特定的内核管理程序去执行,执行完后,回到那条特殊指令后面的一条指令开始执行。称为自愿中断或自陷(Trap)。这些特殊指令称为"访管指令"(访问管理程序)或"自陷指令"(自动调入陷阱),如80x86中的指令"INT n"。

(4) 还有一种情况,既不是外部设备发出,也不是指令本身产生,是在执行指令过程中发生了硬件故障,如电源掉电、线路故障等,使CPU无法继续执行。这类异常是随机发生的,对引起异常的指令的确切位置无法确定,出现这类严重错误时,原程序无法继续执行,只好终止,而由中断服务程序重新启动操作系统。因此这种情况被称为终止(Abort)。

综上所述,上述4种中断源(异常事件)分为两大类。第一种称为外中断(有时简称为中断Interrupt),后面3种称为内中断(也称为异常、程序性中断,或软中断),分别为故障(Fault)、自陷(Trap)和终止(Abort)。内中断是在执行特定的指令时发生的,不需要通过外部中断请求线进行中断请求。

38. 为什么在设计处理器时必须考虑异常和中断的情况?

答:异常和中断是CPU在执行指令过程中,指令自身或外部设备发生的一些特殊事件,这些特殊事件使得当前指令不能继续执行下去,或者,需要CPU停下当前程序的执行,去处理外部中断事件。因此,在数据通路中必须要考虑相应的检测线路,能够发现这种特殊的异常事件,并且还要考虑相应的控制转换电路,能够完成"关中断"、"保护断点"和"将处理异常事件的程序的入口地址加载到PC中",使得控制转到异常处理程序。通常把这种控制转换电路的执行过程称为"中断隐指令"的执行过程,在这个过程中实现了对"内部异常"和"外部中断"的响应。

6.5 单项选择题

1. 机器主频的倒数(一个节拍)等于(　　)。

A. CPU时钟周期　　B. 主板时钟周期
C. 指令周期　　D. 存储周期

2. CPU中控制器的功能是(　　)。

A. 产生时序信号
B. 控制从主存取出一条指令
C. 完成指令操作码译码
D. 完成指令操作码译码,并产生操作控制信号

3. 冯·诺依曼计算机中指令和数据均以二进制形式存放在存储器中,CPU依据(　　)来区分它们。

A. 指令和数据的表示形式不同　　B. 指令和数据的寻址方式不同
C. 指令和数据的访问时点不同　　D. 指令和数据的地址形式不同

4. 下列寄存器中,对汇编语言程序员不透明的是(　　)。

A. 存储器地址寄存器(MAR)　　B. 程序计数器(PC)
C. 存储器数据寄存器(MDR)　　D. 指令寄存器(IR)

5. 下列有关控制器各部件功能的描述中,错误的是(　　)。
   A. 控制单元是其核心部件,用于对指令操作码译码并生成控制信号
   B. PC 称为程序计数器,用于存放将要执行的指令的地址
   C. 通过将 PC 按当前指令长度增量,可实现指令的按序执行
   D. IR 称为指令寄存器,用来存放当前指令的操作码
6. PC 中存放的是后继指令的地址,故 PC 的位数和(　　)的位数相同。
   A. 指令寄存器 IR　　B. 指令译码器 ID
   C. 主存地址寄存器 MAR　　D. 程序状态字寄存器 PSWR
7. 通常情况下,下列(　　)部件不包含在中央处理器(CPU)芯片中。
   A. ALU　　B. 控制器　　C. 寄存器　　D. DRAM
8. 下列有关程序计数器 PC 的叙述中,错误的是(　　)。
   A. 每条指令执行后,PC 的值都会被改变
   B. 指令顺序执行时,PC 的值总是自动加 1
   C. 调用指令执行后,PC 的值一定是被调用过程的入口地址
   D. 无条件转移指令执行后,PC 的值一定是转移目标地址
9. CPU 取出一条指令并执行该指令的时间被称为(　　)。
   A. 时钟周期　　B. CPU 周期　　C. 机器周期　　D. 指令周期
10. 下列有关指令周期的叙述中,错误的是(　　)。
    A. 指令周期的第一个阶段一定是取指令阶段
    B. 乘法指令和加法指令的指令周期总是一样长
    C. 一个指令周期由若干个机器周期或时钟周期组成
    D. 单周期 CPU 中的指令周期就是一个时钟周期
11. 下列有关 CPU 时钟信号的叙述中,错误的是(　　)。
    A. 处理器总是每来一个时钟信号就开始执行一条新的指令
    B. 边沿触发指状态单元总在时钟上升沿或下降沿改变状态
    C. 时钟周期以相邻状态单元之间最长组合逻辑延迟为基准确定
    D. 每个时钟周期称为一个节拍,机器的主频就是时钟周期的倒数
12. 下列有关数据通路的叙述中,错误的是(　　)。
    A. 数据通路由若干操作元件和状态元件连接而成
    B. 数据通路的功能由控制部件送出的控制信号决定
    C. ALU 属于操作元件,用于执行各类算术和逻辑运算
    D. 通用寄存器属于状态元件,但不包含在数据通路中
13. 下列有关取指令操作部件的叙述中,错误的是(　　)。
    A. 取指令操作的延时主要由存储器的取数时间决定
    B. 取指令操作可以和下条指令地址的计算操作同时进行
    C. 单周期数据通路中需用一个指令寄存器存放取出的指令
    D. PC 在单周期数据通路中不需要“写使能”控制信号
14. 下列有关多周期数据通路和单周期数据通路比较的叙述中,错误的是(　　)。
    A. 单周期处理器的 CPI 总比多周期处理器的 CPI 大

B. 单周期处理器的时钟周期比多周期处理器的时钟周期长

C. 在一条指令执行过程中，单周期处理器中的每个控制信号取值一直不变，而多周期处理器中的控制信号可能会发生改变

D. 在一条指令执行过程中，单周期数据通路中的每个部件只能被使用一次，而在多周期中同一个部件可使用多次

15. 下面是有关 MIPS 架构的 R-型指令数据通路设计的叙述：

① 在 R-型指令数据通路中，一定会有一个具有读口和写口的通用寄存器组

② 在 R-型指令数据通路中，一定有一个 ALU 用于对寄存器读出数据进行运算

③ 在 R-型指令数据通路中，一定存在一条路径使 ALU 输出被送到某个寄存器

④ 执行 R-型指令时，通用寄存器堆的“写使能”控制信号一定为“1”

以上叙述中，正确的有(　　)。

A. ①、②、③　　B. ①、②、④　　C. ②、③、④　　D. 全部

16. 下面是有关 MIPS 架构的 lw/sw 指令数据通路设计的叙述：

① 在 lw/sw 指令数据通路中，一定有一个符号扩展部件用于偏移量的扩展

② 在 lw/sw 指令数据通路中，ALU 的控制信号一定为“add”(即 ALU 做加法)

③ 寄存器堆的“写使能”信号在 lw 指令执行时为“1”，在 sw 指令执行时为“0”

④ 数据存储器的“写使能”信号在 lw 指令执行时为“0”，在 sw 指令执行时为“1”

以上叙述中，正确的有(　　)。

A. ①、②、③　　B. ①、②、④　　C. ②、③、④　　D. 全部

17. 下面是有关 MIPS 架构的 beq 指令的单周期数据通路设计的叙述：

① 在 beq 指令的执行过程中，ALU 的两个输入都来自寄存器堆

② 在 beq 指令数据通路中，ALU 的控制信号一定为“sub”(即 ALU 做减法)

③ 在 beq 指令数据通路中，一定有一个加法器用于计算目标转移地址

④ 在 beq 指令的执行过程中，数据不会流经符号扩展部件

以上叙述中，正确的有(　　)。

A. ①、②、③　　B. ①、②、④　　C. ②、③、④　　D. 全部

18. 某计算机指令集中包含有 RR 型运算指令、取数指令 Load、存数指令 Store、分支指令 Branch 和跳转指令 Jump。若采用单周期数据通路实现该指令系统，各主要功能部件的操作时间为：指令存储器和数据存储器都是 3ns；ALU 和加法器都是 2ns；寄存器堆的读和写都是 1ns。在不考虑多路复用器、控制单元、PC、符号扩展单元和传输线路等延迟的情况下，该计算机时钟周期至少为(　　)。

A. 6ns　　B. 8ns　　C. 10ns　　D. 12ns

19. 下列有关微指令格式的描述中，错误的是(　　)。

A. 相对于直接控制法(不译法)，字段直接编码法的控存利用率更高

B. 相对于字段直接编码法，直接控制法(不译法)的执行速度更快

C. 相对于断定法(下址字段法)，采用计数器法的微指令格式更短

D. 相对于水平型微指令，一条垂直型微指令中包含的微命令更多

20. 下列有关指令和微指令之间关系的描述中，正确的是(　　)。

A. 一条指令的功能通过执行一条微指令来实现

B. 一条指令的功能通过执行一个微程序来实现

C. 一条微指令的功能通过执行一条指令来实现

D. 一条微指令的功能通过执行一个微程序来实现

21. 相对于微程序控制器,硬布线控制器的特点是(　　)。

A. 指令执行速度慢,指令功能的修改和扩展容易

B. 指令执行速度慢,指令功能的修改和扩展难

C. 指令执行速度快,指令功能的修改和扩展容易

D. 指令执行速度快,指令功能的修改和扩展难

22. 以下给出的事件中,无须异常处理程序进行处理的是(　　)。

A. 缺页故障　　B. 访问 cache 缺失

C. 地址越界　　D. 除数为 0

23. 以下有关"自陷"(Trap)异常的叙述中,错误的是(　　)。

A. "自陷"是人为预先设定的一种特定处理事件

B. 可由"访管指令"或"自陷指令"的执行进入"自陷"

C. 一定是出现了某种异常情况才会发生"自陷"

D. "自陷"发生后 CPU 将进入操作系统内核程序执行

【参考答案】

1. A	2. D	3. C	4. B	5. D	6. C	7. D
8. B	9. D	10. B	11. A	12. D	13. C	14. A
15. D	16. D	17. A	18. C	19. D	20. B	21. D
22. B	23. C					

6.6 分析应用题

1. 某计算机主频为 800MHz,其 CPU 采用三级时序(机器周期-节拍-脉冲)进行定时,为单脉冲节拍方式,每个机器周期的基本宽度为 4 个节拍。该机每个指令周期平均有 5 个机器周期,并平均访问 2 次主存,采用非缓存(无 cache)访问。请回答下列问题:

(1) 若采用异步方式访问内存,每个"存储器读"机器周期平均需 8 个节拍,每个"存储器写"机器周期平均需 6 个节拍,则执行一条指令的平均时间为多少? MIPS 数为多少? 平均 CPI 为多少?

(2) 若采用同步方式访问内存,每个"存储器读"机器周期需插入 4 个"等待"状态,每个"存储器写"机器周期无须"等待"状态,则执行一条指令的平均时间为多少? MIPS 数为多少? 平均 CPI 为多少?(提示:一个"等待状态"即一个节拍)

【分析解答】

早期的计算机中没有 cache,CPU 访存时,由于主存速度慢,无法像访问寄存器和 cache 那样能在一个节拍内存取好数据,因此,需要 CPU 等待若干个节拍才能完成存储访问。因而"存储器读"和"存储器写"机器周期比基本的无访存操作机器周期长。

(1) CPU 和主存之间采用异步方式通信时,CPU 每次发送读或写命令后,便在随后的每个节拍内采样主存送来的"存储操作完成"(MFC)信号,以便在主存数据准备好时,到存

储器数据寄存器(MDR)中取数据或将 MDR 中数据写到主存。由题意可知,在异步方式下,每条指令的平均时钟周期(节拍)数为$(5-2)\times4+2\times((8+6)/2)=26$,所以 CPI=26。执行一条指令的平均时间为$26\times1/(800\times10^6\,\text{Hz})=32.5\text{ns}$。MIPS 数约为 800/26=31。

(2) CPU 和主存之间采用同步方式通信时,CPU 每次发送读或写命令后,便按照同步通信协议,在等待一个固定长度的时间(若干个节拍)后,到存储器数据寄存器(MDR)中取数据或将 MDR 中数据写到主存。由题意可知,在同步方式下,每条指令的平均时钟周期(节拍)数为$(5-2)\times4+2\times((4+4+4)/2)=24$,所以 CPI=24。执行一条指令的平均时间为 $24\times1/(800\times10^6\,\text{Hz})=30\text{ns}$。MIPS 数约为 800/24=33。

2. 某计算机的字长为 16 位,采用 16 位定长指令字格式,其部分数据通路的结构如图 6.8 所示。假设MAR 的输出一直处于使能状态。加法指令“add (R1),R0”的功能为 M[R[R1]]←M[R[R1]]+R[R0]。

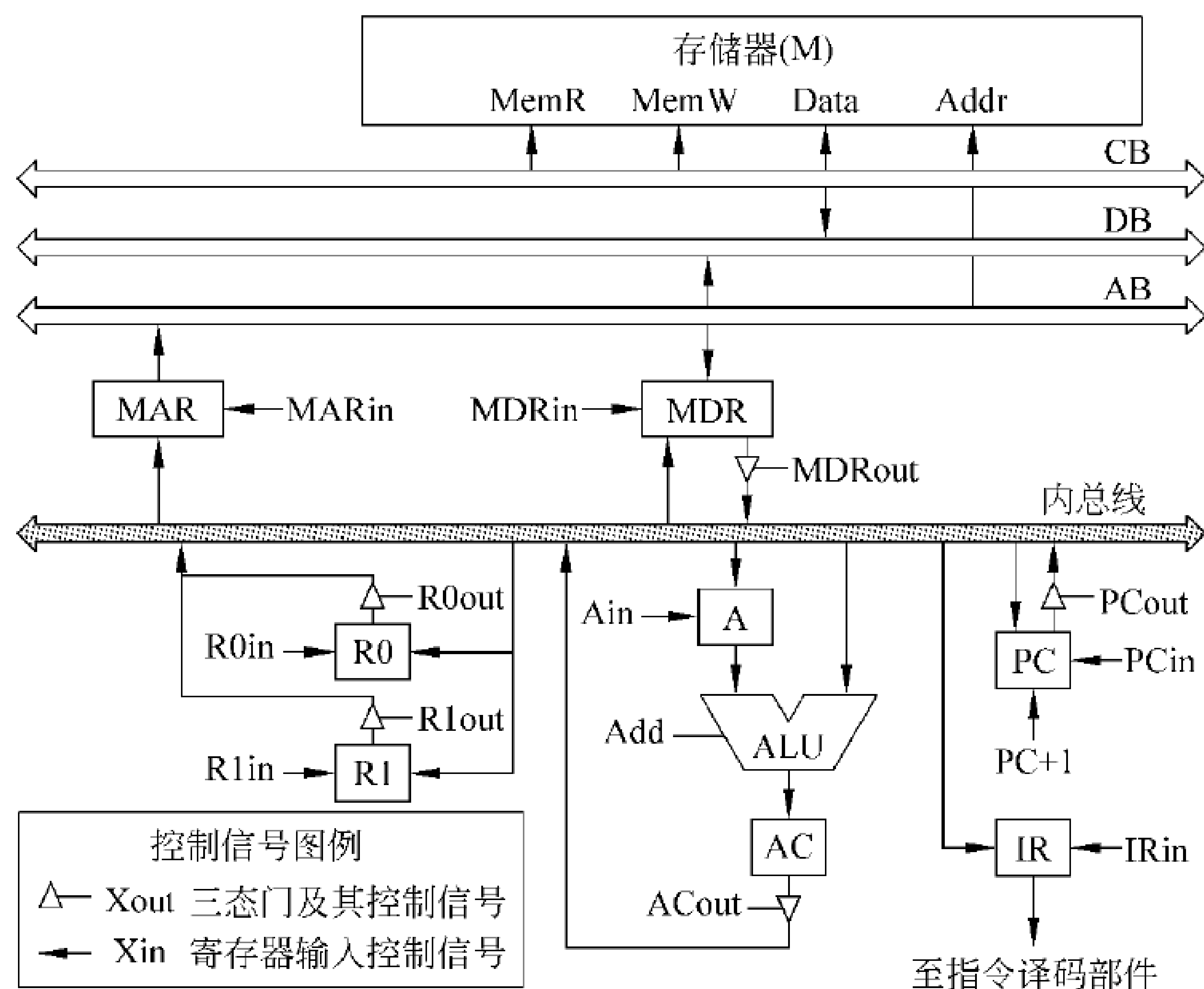


图 6.8 题 2 的数据通路

表 6.2 给出了上述指令取指和译码阶段每个节拍(时钟周期)的功能和有效控制信号,请按表中描述方式列出指令执行阶段每个节拍的功能和有效控制信号,并说明需要多少节拍。

表 6.2 题 2 取指令阶段的控制信号

时钟	功 能	有效控制信号
C1	MAR←(PC)	PCout,MARin
C2	MDR←M(MAR) PC←(PC)+1	MemR PC+1
C3	IR←(MDR)	MDRout,IRin
C4	指令译码	无

【分析解答】

加法指令“add (R1),R0”的执行阶段每个节拍的功能和控制信号如表 6.3 所示。

表 6.3　题 2 指令执行阶段的控制信号

时钟	功　　能	有效控制信号
C5	MAR←(R1)	R1out,MARin
C6	MDR←M(MAR) A←(R0)	MemR R0out,Ain
C7	AC←A+(MDR)	MDRout,Add
C8	MDR←(AC)	ACout,MDRin
C9	M(MAR)←MDR	MemW

从表 6.3 可以看出，在 C6 节拍中同时进行了存储器读和寄存器间传送，这样，该指令的执行阶段共有 5 个节拍。当然，也可将存储器读和寄存器间传送操作安排在不同的节拍内进行，这样得到表 6.4。此时，执行阶段共有 6 个节拍。

表 6.4　题 2 指令执行阶段的控制信号

时钟	功　　能	有效控制信号
C5	MAR←(R1)	R1out,MARin
C6	MDR←M(MAR)	MemR
C7	A←(MDR)	MDRout,Ain
C8	AC←A+(R0)	R0out,Add
C9	MDR←(AC)	ACout,MDRin
C10	M(MAR)←MDR	MemW

3. 假定在如图 6.9 所示的单总线数据通路中，总线传输延迟和 ALU 运算时间分别是 20ps 和 200ps，寄存器建立时间为 10ps，寄存器保持时间为 5ps，寄存器的锁存延迟(Clk-to-Q)为 4ps，控制信号的生成延迟(Clk-to-signal)为 7ps，三态门接通时间为 3ps，则从当前时钟到达开始算起，完成以下操作的最短时间是多少？各需要几个时钟周期？

(1) 将数据从一个寄存器传送到另一个寄存器。

(2) 将程序计数器 PC 加 1。

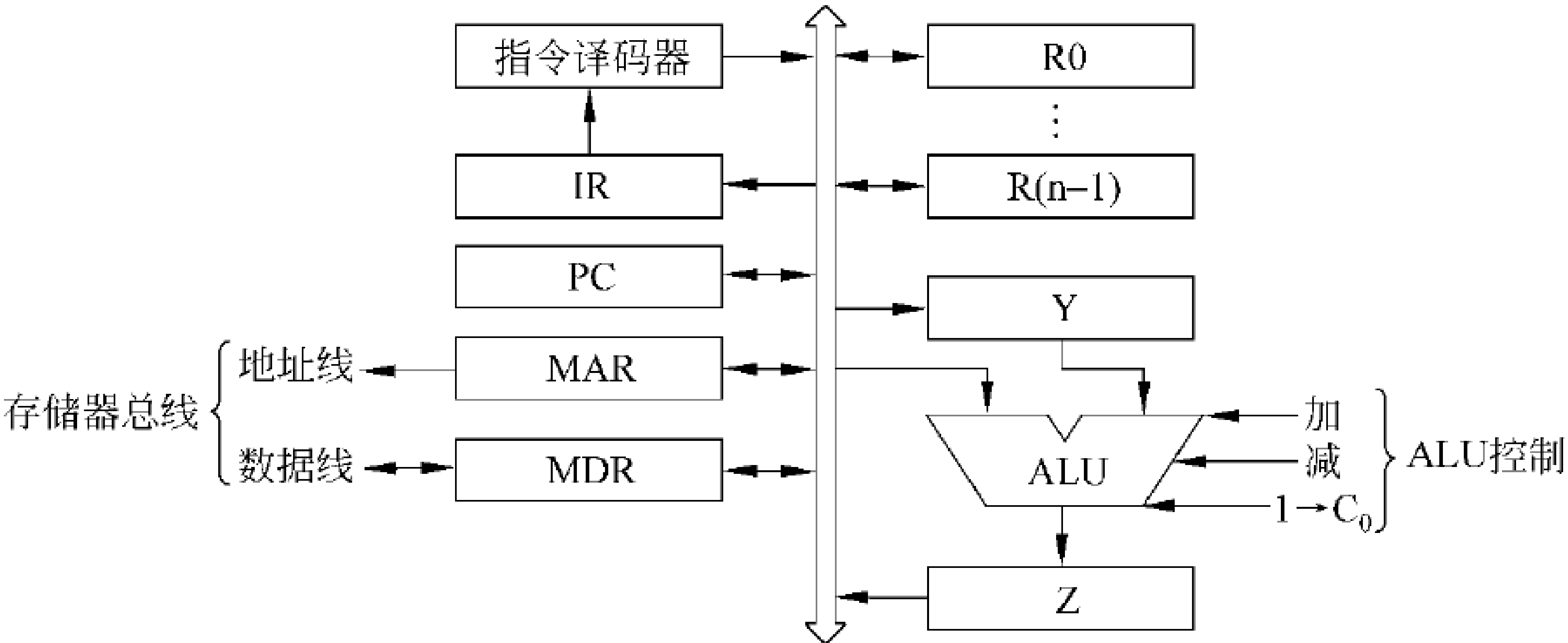


图 6.9　单总线数据通路

【分析解答】

图 6.9 所示的数据通路中，所有与内部总线相连的寄存器都有相应的 R_{in} 和/或 R_{out} 控制信号，以控制总线和寄存器之间的数据传送。总线和 ALU 输入端之间、Y 寄存器与 ALU 输入端之间都无须控制信号。ALU 输出与 Z 寄存器之间可以有控制信号 Z_{in}，也可以没有。若没有 Z_{in}，则每来一个时钟，ALU 的输出总是被写入 Z 寄存器。以下说明中，为了明显表示 ALU 输出送 Z 寄存器，假定有控制信号 Z_{in}。

图 6.10 给出了单总线数据通路中主要路径的定时。时钟边沿到达后，经过 Clk-to-Q 的延时，寄存器中的内容被读出，同时，在指令译码器中的控制逻辑生成当前时钟周期内所需要的控制信号，其延时为 Clk-to-signal。随后，由生成的控制信号 Ri_{out} 接通三态门，并使寄存器数据在总线上传输。若当前时钟周期内不做 ALU 运算而是直接在寄存器之间传送，则总线上的数据在 Rj_{in} 的控制下直接被送到目的寄存器 Rj 的输入端，在下一个时钟边沿到来之前，总线上的数据必须继续稳定一段寄存器建立时间，以使寄存器 Rj 的输入在这段建立时间内保持不变，如图 6.10(a)所示；若当前时钟周期内有 ALU 运算，则还需 ALU 电路延时，最后 ALU 结果直接送 Z 寄存器，在下一个时钟边沿到来之前，ALU 的输出必须继续稳定一段寄存器建立时间，以使寄存器 Z 的输入在这段建立时间内保持不变，如图 6.10(b)所示。

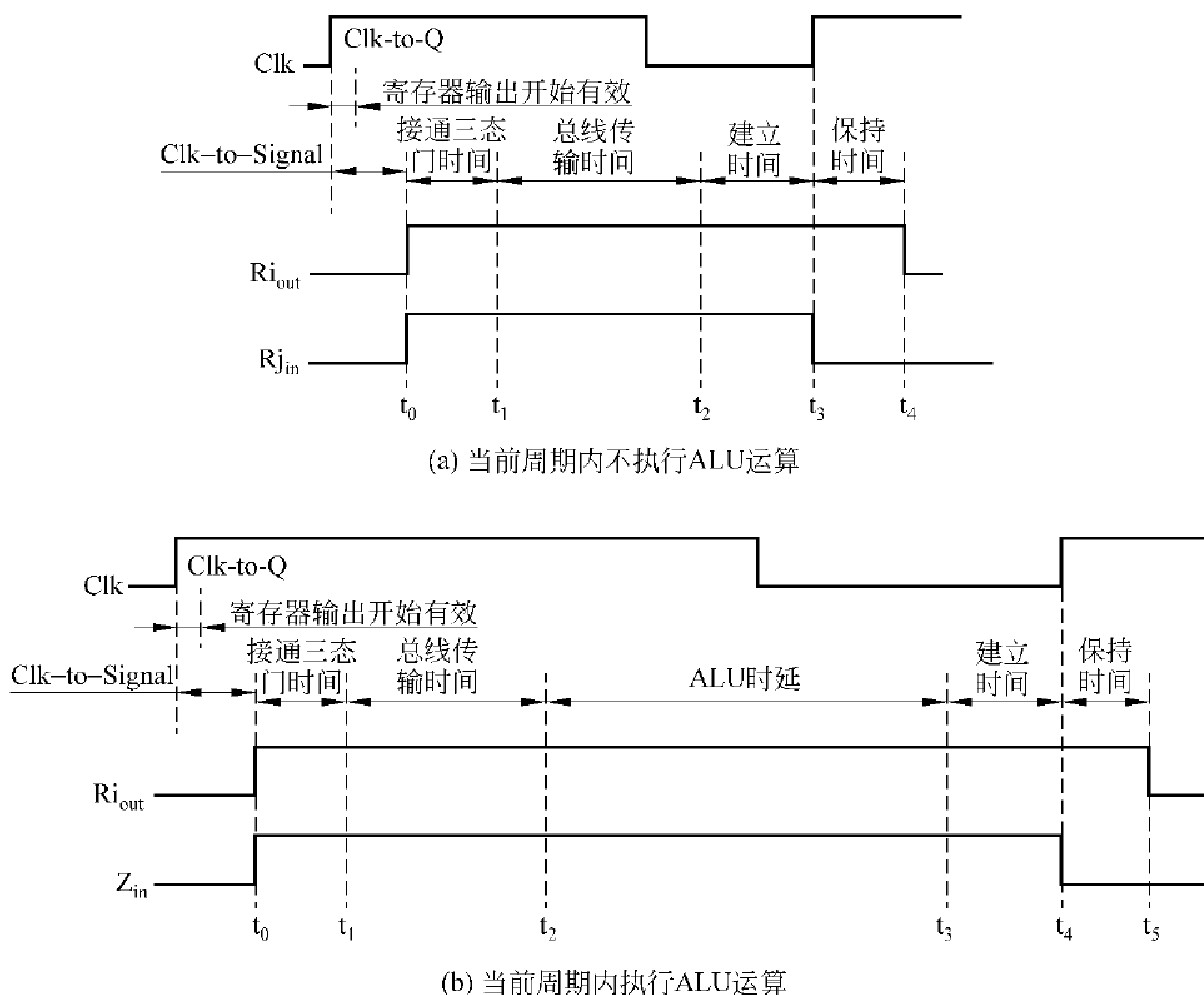


图 6.10 单总线数据通路中主要路径的定时

(1) 由图 6.10(a)可知，寄存器之间进行传送的时间延迟至少为 7＋3＋20＋10＝40ps。在寄存器数据传送过程中，只需要在一个寄存器中保存信息，因此只需要一个时钟周期就可完成该操作。

(2) 将 PC 中的内容加 1 送 PC，被分解成以下两个过程：PC 加 1 送 Z、Z 送 PC。对于第一个过程，由图 6.10(b)可知，其延迟至少为 7+3+20+200+10=240ps；第二个过程实现的是寄存器之间的传送，因此延迟至少为 40ps。因为在该操作过程中分别在寄存器 Z 和 PC 中保存了共两次信息，所以需要两个时钟周期才能完成该操作。

4. 假定某计算机字长 16 位，CPU 内部结构如图 6.9 所示，CPU 和存储器之间采用同步方式通信，按字编址。采用定长指令字格式，指令由两个字组成，第一字指明操作码、寻址方式和一个寄存器编号，第二字为立即数 imm16。若一次存储访问所用时间为 2 个 CPU 时钟周期 Read1 和 Read2，每次存储访问存取一个字，取指令阶段第二次访存已将 imm16 取到 MDR 中，请写出下列指令在执行阶段的控制信号序列，并说明需要几个时钟周期。

(1) 将 imm16 加到寄存器 R1 中，即 R[R1]←R[R1]+imm16。

(2) 将存储单元 imm16 中的内容加到寄存器 R1 中，此时，imm16 为直接地址。即 R[R1]←R[R1]+M[imm16]。

(3) 将存储单元 imm16 中的内容作为地址访问主存，将读出的内容再作为地址访问主存，然后将读出的内容加到寄存器 R1 中。此时，imm16 为间接地址。即 R[R1]←R[R1]+M[M[imm16]]。

【分析解答】

图 6.9 所示的数据通路中，所有与内部总线相连的寄存器都有相应的 R_{in} 和/或 R_{out} 控制信号，以控制总线和寄存器之间的数据传送。总线和 ALU 输入端之间、Y 寄存器与 ALU 输入端之间都无须控制信号。ALU 输出与 Z 寄存器之间可以有控制信号 Z_{in}，也可以没有，此时，每来一个时钟，ALU 的输出总是被写入 Z 寄存器。以下说明中，为了明显表示 ALU 输出送 Z 寄存器，假定有控制信号 Z_{in}。

(1) 指令功能为 R[R1]←R[R1]+imm16 时，执行阶段不需要访存操作，因此，可用 3 个时钟周期完成，分别包含以下各控制信号。

MDR_{out}, Y_{in}
$R1_{out}$, add, Z_{in}
Z_{out}, $R1_{in}$

(2) 指令功能为 R[R1]←R[R1]+M[imm16]时，执行阶段需要一次访存操作，因此，至少需要以下 5 个时钟周期。其中 $R1_{out}$、Y_{in} 这两个控制信号可以在 Read1 周期就送出，并在 Read2 周期中保持不变，也可以到 Read2 周期时再送出。

MDR_{out}, MAR_{in}
Read1, ($R1_{out}$, Y_{in})
Read2, $R1_{out}$, Y_{in}
MDR_{out}, add, Z_{in}
Z_{out}, $R1_{in}$

(3) 指令功能为 R[R1]←R[R1]+M[M[imm16]]时，执行阶段需要两次访存操作，因此，至少需要以下 8 个时钟周期。对 $R1_{out}$、Y_{in} 这两个控制信号的处理同(2)中一样。

MDR_{out}, MAR_{in}
Read1

```
Read2
MDRout, MARin
Read1, (R1out, Yin)
Read2, R1out, Yin
MDRout, add, Zin
Zout, R1in
```

5. 图 6.11 给出了某 CPU 内部结构的一部分，MAR 和 MDR 直接连到存储器总线（图中省略）。在两个总线 A 和 B 之间的所有数据传送都需经过算术逻辑部件 ALU。ALU 的部分功能及其控制信号如下。

```
MOVa: F=A;     MOVb: F=B;
a+1: F=A+1;    b+1: F=B+1
a-1: F=A-1;    b-1: F= B-1
```

其中 A 和 B 是 ALU 的输入，F 是 ALU 的输出。假定调用指令 Call 占两个字，第一个字是操作码，第二个字给出子程序的起始地址，返回地址保存在主存的栈中，用 SP（栈指示器）指向栈顶，主存按字编址，每次以同步方式从主存读取一个字。请写出读取并执行 Call 指令所要求的控制信号序列（提示：当前指令地址在 PC 中），并说明至少需要多少个时钟周期。

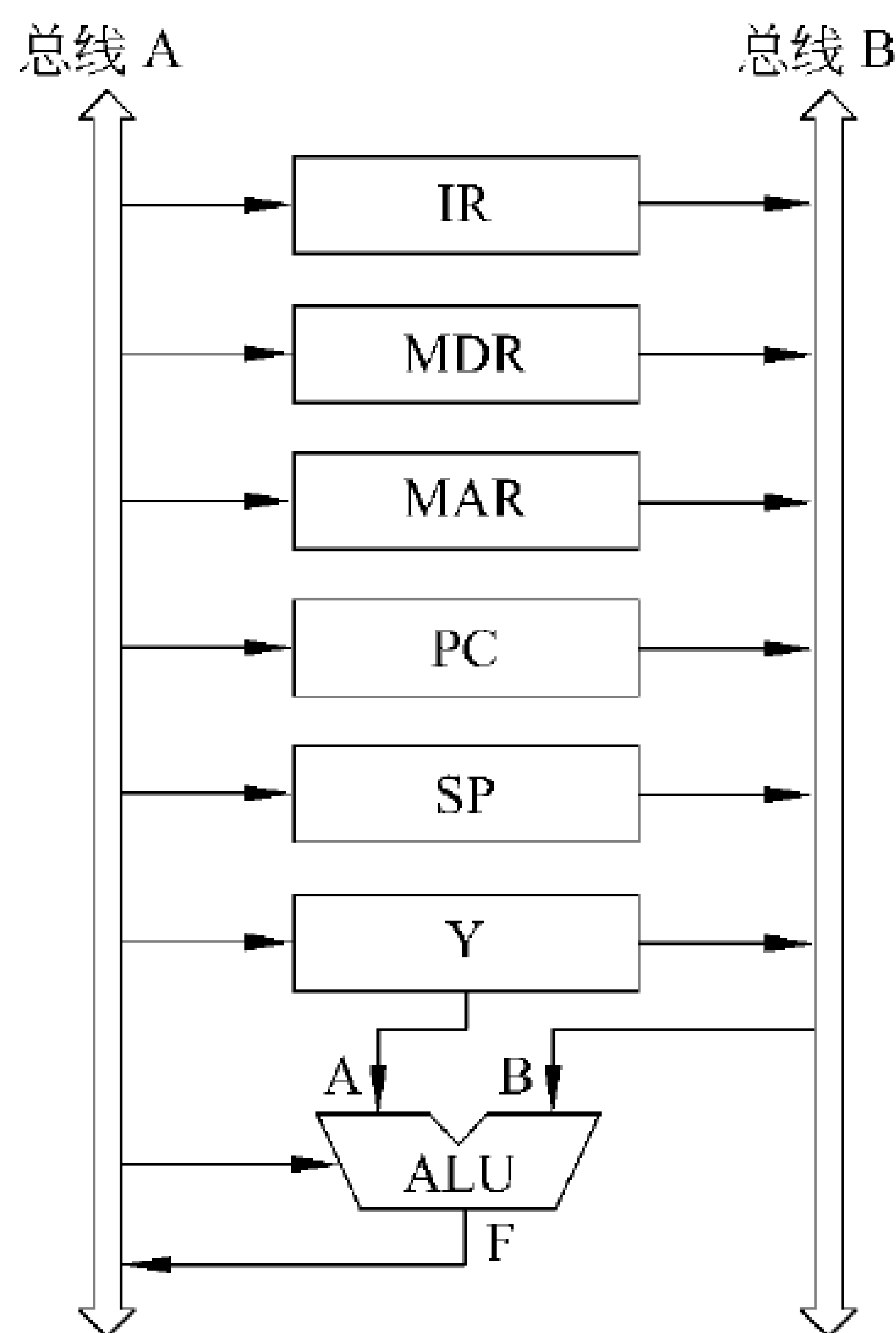


图 6.11　题 5 的图示

【分析解答】

因为采用同步方式读写内存，所以在 Read 和 Write 信号后不需加等待信号 WMFC。Call 指令占两个字，主存按字编址，每次从主存读取一个字，因此，Call 指令需要读两次主存每次 PC 加 1。其指令周期分为以下 3 个阶段。

(1) 读取指令操作码：将 PC 的内容作为地址访问存储器，取出指令的操作码，送指令寄存器 IR，同时 PC+1 送 PC，以指向指令的第二个字。至少需要 3 个时钟周期（节拍）。

```
PCout, MOVb, MARin
Read, b+1, PCin
MDRout, MOVb, IRin
```

(2) 读取子程序首地址：将 PC 的内容作为地址，取出指令的第二个字（即子程序入口地址）送 PC，以使下一个指令周期从子程序的第一条指令开始执行。同时，计算 PC+1 以得到返回地址，并将返回地址送 Y 寄存器。至少需要 3 个时钟周期。

```
PCout, MOVb, MARin
Read, b+1, Yin
MDRout, MOVb, PCin
```

(3) 保存返回地址至栈中：将临时保存在 Y 寄存器的返回地址送到栈顶保存，并将 SP 的内容减 1，以自动调整栈顶指针。至少需要 3 个时钟周期。

```
SPout, MOVb, MARin
```

Y_{out}, MOVb, MDR_{in}

Write, SP_{out}, b-1, SP_{in}

显然，上述每个节拍中执行的操作所需要的时间不等，其中，存储访问(Read/Write)时间最长，时钟周期以最长的存储访问时间为准，Call 指令的指令周期至少有 9 个时钟周期(节拍)。

如果将第一次 PC+1 的结果送到 Y 寄存器，第二阶段以 Y 的内容作为地址访问主存，并继续对 Y 寄存器加 1，结果送 PC，则也能实现 Call 指令的功能。这种方式下，也是 9 个时钟周期。

6. 某计算机字长 16 位，标志寄存器 Flag 中的 ZF、SF 和 OF 分别是零标志、符号标志和溢出标志，采用双字节定长指令字。假定 bgt (大于零转移) 指令的第一个字节指明操作码 OP 和寻址方式 MOD，第二个字节为偏移地址 imm8，用补码表示。指令功能是：

若$(ZF+(SF \oplus OF)=0)$ 则 $PC=PC+2+imm8\times 2$，否则 $PC=PC+2$

请回答下列问题或完成相应任务。

(1) 该计算机的编址单位是多少?

(2) bgt 指令执行的是带符号整数比较还是无符号数比较? 偏移地址 imm8 的含义是什么? 转移目标地址的范围是什么?

(3) 画出实现 bgt 指令的数据通路。

【分析解答】

(1) 因为 PC 的增量是 2，且每条指令占 2 个字节，所以编址单位是字节。

(2) 根据“大于”条件判断表达式，可以看出该 bgt 指令实现的是带符号整数比较。因为无符号数比较时，其判断表达式中没有溢出标志 OF。偏移地址 imm8 为补码表示，说明转移目标指令可能在 bgt 指令之前，也可能在 bgt 指令之后。计算转移目标地址时，偏移量为 imm8×2，说明 imm8 不是相对地址，而是相对指令条数。imm8 的范围为−128~127，故转移目标地址的范围是 PC+2+(−128×2)~PC+2+127×2，即转移目标地址的范围是相对于 bgt 指令的前 254 个单元到后 256 个单元之间，用指令条数来衡量的话，就是相对于 bgt 指令的前 127 条指令到后 128 条指令之间。

(3) 实现 bgt 指令的数据通路如图 6.12 所示。

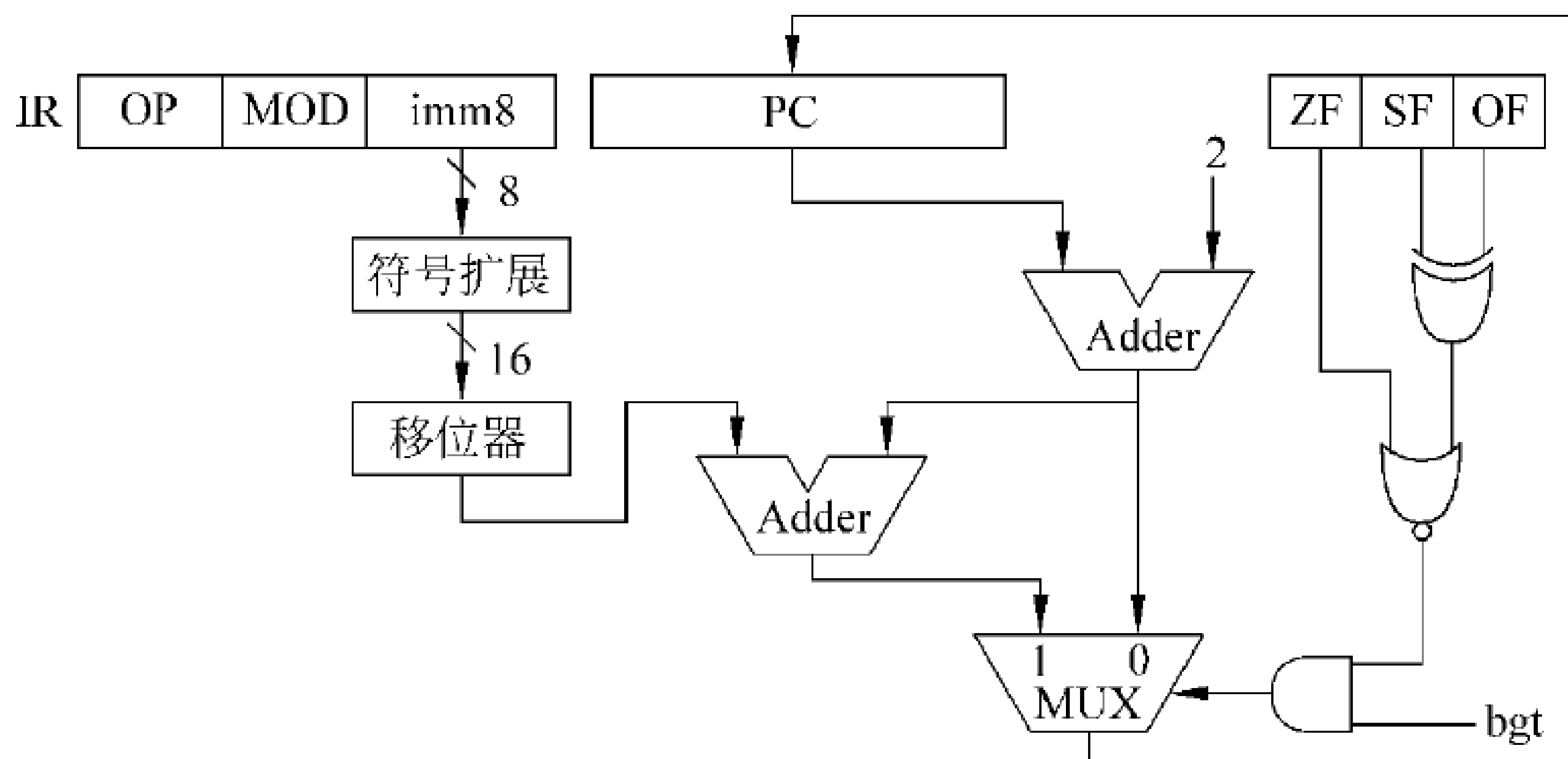


图 6.12　实现 bgt 指令的数据通路

7. 如果图 6.4 给出的单周期数据通路对应的控制逻辑发生错误，使得控制信号

RegWr、RegDst、Branch、MemWr、ExtOp 中某一个在任何情况下总是为 0,则这些控制信号分别为 0 时哪些指令不能正确执行? 要求分别讨论。

【分析解答】

若 RegWr=0,则所有需写结果到寄存器的指令(如: R-型指令、Load 类指令等)都不能正确执行,因为寄存器不发生写操作。

若 RegDst=0,则所有 R-型指令都不能正确执行,因为目的寄存器指定为 Rt 而不是 Rd。

若 Branch=0,则 Branch 类指令可能出错,因为永远不会发生转移。

若 MemWr=0,则 Store 类指令不能正确执行,因为存储器不能写入所需数据。

若 ExtOp=0,则需要符号扩展的指令(如 beq、lw/sw 等)发生错误,因为进行的是 0 扩展。

8. 如果图 6.4 给出的单周期数据通路对应的控制逻辑发生错误,使得控制信号 RegWr、RegDst、Branch、MemWr、ExtOp 中某一个在任何情况下总是为 1,则这些控制信号分别为 1 时哪些指令不能正确执行? 要求分别讨论。

【分析解答】

若 RegWr=1,则所有不需写结果到寄存器的指令(如 sw、beq 等)都不能正确执行,因为错误地将某个值写入了某个寄存器。

若 RegDst=1,则 lw 等部分 I-型指令不能正确执行,因为目的寄存器指定为 Rd 而不是 Rt。

若 Branch=1,则非 Branch 类指令可能出错,因为可能会发生不必要的转移。

若 MemWr=1,则除 Store 类指令外其他指令都不能正确执行,因为会写入数据到存储器。

若 ExtOp=1,则需要 0 扩展的指令(如 ori 等)可能会发生错误,因为进行的是符号扩展。

9. 如果图 6.6 给出的多周期数据通路对应的控制逻辑发生错误,使得控制信号 PCWr、IRWr、RegWr、BrWr、PCSource、MemWr、MemtoReg、PCWrCond、R-type 中某一个在任何情况下总是为 0,则这些控制信号分别为 0 时哪些指令不能正确执行? 要求分别讨论。

【分析解答】

若 PCWr=0,则所有指令都不能正确执行,因为无法正确地更新 PC。

若 IRWr=0,则所有指令都不能正确执行,因为 IR 中不能写入当前指令,也就无法进行正确的指令译码,因此后续的指令执行过程都不正确。

若 RegWr=0,则所有需要写结果到寄存器的指令(如 R-型指令、lw 等部分 I-型指令)都不能正确执行,因为寄存器不发生写操作。

若 PCSource=00,则除 j 指令之外的其他指令都不能正确得到下条指令地址。

若 MemWr=0,则所有 Store 类指令执行错误,因为数据不能写入存储器中。

若 MemtoReg=0,则所有 Load 类指令执行错误,因为写入寄存器的是 ALU 输出而不是读出的存储单元的内容。

若 PCWrCond=0,则 Branch 类指令执行可能出错,因为永远不会发生转移。

若 R-type=0,则所有 R-型指令的执行可能出错,因为控制信号 ALUctr 的取值不是来自对 R-型指令进行解释的 ALU 局部控制器。

10. 如果图 6.6 给出的多周期数据通路对应的控制逻辑发生错误,使得控制信号 PCWr、IRWr、RegWr、BrWr、PCSource、MemWr、MemtoReg、PCWrCond、R-type 中某一个在任何情况下总是为 1,则这些控制信号分别为 1 时哪些指令不能正确执行? 要求分别讨论。

【分析解答】

若 PCWr=1,则程序执行顺序失控,因为每个时钟都会更新 PC。

若 IRWr=1,则所有指令都可能出错,因为每个时钟周期都会写入 IR,因此写入的不是当前指令。

若 RegWr=1,则所有不需写结果到寄存器的指令(如 sw、beq 等)都不能正确执行,因为错误地将某个值写入了某个寄存器。

若 PCSource=01,则 j 指令和 Branch 类指令不能正确得到下条指令地址。

若 MemWr=1,则除 Store 类指令外的所有指令执行错误,因为在存储器写入了不该写的信息。

若 MemtoReg=1,则除 Load 类指令外的所有指令执行错误,因为选择了错误的信息写入寄存器。

若 PCWrCond=1,则除 Branch 类指令外的所有指令可能出错,因为可能会发生不必要的转移。

若 R-type=1,则所有非 R-型指令的执行可能出错,因为控制信号 ALUctr 的取值来自对 R-型指令进行解释的 ALU 局部控制器。

11. swap 指令的功能是实现两个寄存器内容的互换。在 MIPS 指令集中增加一条 swap 指令有两种方式,一种是采用伪指令(软件)方式,这种情况下,当执行到 swap 指令时,用若干条已有指令构成的指令序列来代替实现;另一种做法是改动硬件以实现 swap 指令,这种情况下,当执行到 swap 指令时,则直接在 swap 指令对应的数据通路(硬件)上执行。

(1) 写出 MIPS 中用伪指令方式实现“swap rs, rt”时的指令序列。

(2) 假定用硬件的实现 swap 指令时会使每条指令的执行时间增加 10%,则 swap 指令在程序中占多大的比例才值得用硬件方式来实现?

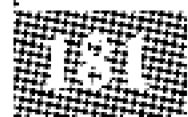
(3) 采用硬件方式实现时,在不对寄存器堆进行修改的情况下,能否在单周期数据通路中实现 swap 指令? 对于多周期数据通路的情况又怎样?

【分析解答】

(1) 若在伪指令“swap rs, rt”的指令序列中使用除 rs 和 rt 以外的额外寄存器,则额外寄存器的内容会被指令序列破坏,因此,伪指令的指令序列中一般不能用额外寄存器。伪指令“swap rs, rt”的指令序列包含以下 3 条指令,没有用到额外寄存器。

```
xor  rs, rs, rt
xor  rt, rs, rt
xor  rs, rs, rt
```

(2) 假定“swap rs, rt”指令所占比例为 $x(0\leqslant x\leqslant 1)$,其他指令比例为 $1-x$,则用硬件



实现该指令时，程序执行时间为原来的 $1.1\times(x+1-x)=1.1$ 倍。用软件实现该指令时，程序执行时间为原来的 $3x+1-x=2x+1$ 倍。因此，当 $1.1<2x+1$ 时，硬件的实现才有意义，由此可知，$x>0.05$，即当“swap rs, rt”指令在程序中的比例大于5%时，才值得用硬件方式来实现该指令。

(3) 在单周期数据通路中，所有指令的执行都在一个时钟周期内完成，数据总是在时钟边沿被写入寄存器堆，即本条指令执行的结果总是在下条指令开始(即下个时钟到来)时，才开始被写到寄存器堆中，因此一个时钟周期只能写一次寄存器。而 swap 指令执行过程中需要多次写寄存器，因此，在不对寄存器堆进行修改的情况下，无法在单周期数据通路中实现 swap 指令；对于多周期数据通路，一个指令周期可以有多个时钟周期，因而可以多次写寄存器，因此，在不对寄存器堆进行修改的情况下，swap 指令可以在多周期数据通路中实现。

12. 已知 MIPS 中有一条转移指令 jr(Jump Register)，其指令格式如下：

31～26	25～21	20～16	15～11	10～6	5～0
000000	rs	0	0	0	001000

其功能是读出 rs 寄存器的内容送 PC。若使图 6.4 所示的单周期数据通路也能执行 jr 指令，则如何修改单周期数据通路？需要增加什么控制信号？

【分析解答】

jr 指令采用 R-型指令格式，其中的 rt 和 rd 字段都是 0，即 rt=rd=$0。因为 MIPS 处理器中寄存器 $0 的内容永远是全 0，它可被读出，但不能改变其值，所以，写入信号对寄存器 $0 不起作用，因而，可以把 jr 指令看成是结果送 PC 的加法指令，其功能为 R[rd]←R[rs]+R[rt]，PC←R[rs]+R[rt]。因此，原 R-型指令的数据通路不需要改动，而只要增加 ALU 结果送 PC 的数据通路，并对控制信号作相应修改即可。具体修改如下。

(1) PC 的输入端再增加一个来源：ALU 输出，因此，在 PC 输入端需再加一个多路选择器 MUX，原先的输入作为 MUX 其中的一个输入，另一个输入为 ALU 输出。

(2) 增加一个控制信号 JReturn，用于对新加 MUX 进行控制，执行 jr 指令时，JReturn=1，选择 ALU 输出送 PC；执行其他指令时，JReturn=0，选择原来的输入值送 PC。

(3) ALU 控制器中增加 func=001000B 时相应的译码输出，其输出控制信号取值为 ALUctr=add。

13. 假设 MIPS 指令系统中有一条 I-型带符号整数比较指令“bgt rs, rt, imm16”，其功能为：若 R[rs]>R[rt] 则 PC=PC+4+imm16×4，否则 PC=PC+4。

若 ALU 能产生 ZF(零)、SF(符号)和 OF(溢出)三个标志的输出，请在图 6.6 所示的多周期数据通路中增加实现 bgt 指令的数据通路，并给出指令在执行周期中相应的控制信号取值。

【分析解答】

比较指令需要对两个寄存器 rs 和 rt 中的内容做减法运算(用 ALU 中的加法器实现)，ALU 根据运算结果得到 ZF、SF 和 OF 三个标志信息。在此，rs 和 rt 中的内容被看成带符号整数，因此，判断 R[rs]>R[rt]的条件表达式为 $\overline{ZF}\cdot\overline{(SF\oplus OF)}$。图 6.6 中的多周期数据通路已能支持 beq 指令的执行，而 beq 指令和 bgt 指令的目标转移地址是一样的，因此，

只要在图 6.6 中原先 beq 指令的条件判断逻辑基础上增加 bgt 指令的条件判断逻辑即可，修改后的数据通路如图 6.13 所示，其中带阴影部分是修改或增加的数据通路。

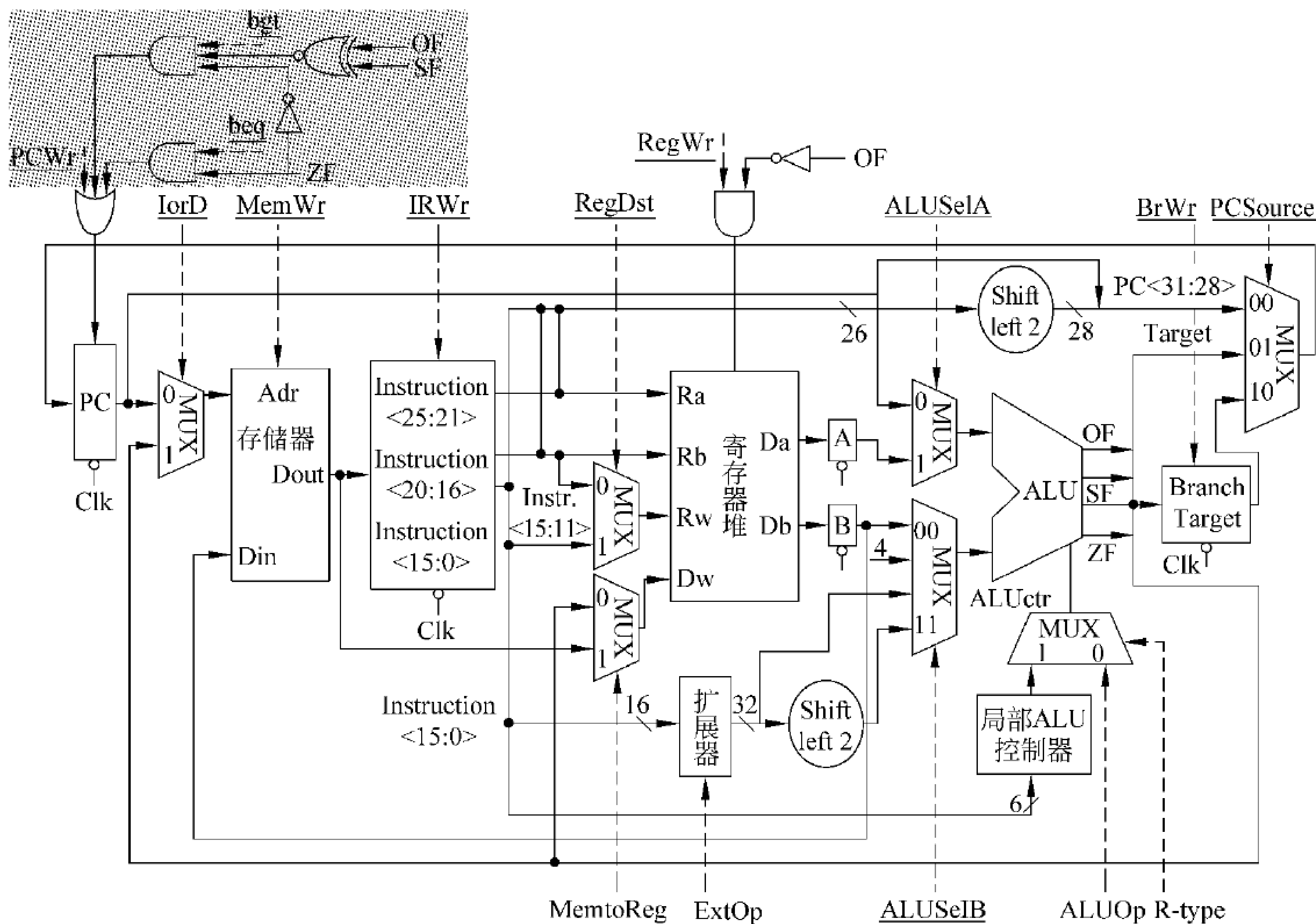


图 6.13 增加了 bgt 指令的多周期数据通路

在多周期数据通路中，bgt 指令与 beq 指令一样，都需要 3 个时钟周期，前两个时钟周期为取指周期和译码/取数周期，它们在所有指令的执行过程中都是一样的，第 3 个周期为执行周期，在此周期中，各控制信号的取值为 beq＝PCWr＝MemWr＝IRWr＝RegWr＝BrWr＝R-type＝0，bgt＝ALUSelA＝1，ALUSelB＝00，ALUOp＝sub，PCSource＝10，其余任意。

14. 已知 MIPS 中有一条 lui(Load Upper Immediate)指令，其指令格式如下：

31　26	25　21	20　16	15　0
001111	00000	rt	imm16

其功能是将立即数 imm16 送到 rt 寄存器的高 16 位，低 16 位补 0。若使图 6.6 所示的多周期数据通路也能执行 lui 指令，则如何修改该多周期数据通路？需要增加什么控制信号？该指令对应的时钟周期数是多少？除取指周期和译码/取数周期外的其他周期中控制信号的取值是什么？

【分析解答】

lui 指令采用 I-型指令格式，其中 rs＝ $0。该指令的功能可以描述成 R[rt]← R[$0]＋imm16×2^{16}。因此，只要在图 6.6 中对原扩展器稍加修改，然后借助 I-型运算类指令数据通路就可实现 lui 指令。具体修改如下。

(1) 修改原扩展器,使其具有高位零扩展、高位符号扩展和低位零扩展3种扩展功能。

(2) 控制信号 ExtOp 从原来的1位扩展为2位,用于控制扩展器进行3种扩展操作。可以定义 ExtOp=01 时进行高位零扩展;ExtOp=10 时进行高位符号扩展;ExtOp=11 时进行低位零扩展。

该指令的执行过程同I-型运算类指令,因此,与 ori 指令类似,其时钟周期数为4。第3个时钟周期各控制信号取值为 ALUSelA=1,ALUSelB=10,ALUOp=add,ExtOp=11,MemtoReg=RegDst=RegWr=PCWr=PCWrCond=IRWr=MemWr=BrWr=R-type=0,其余任意;第4个时钟周期各控制信号取值为 ALUSelA=RegWr=1,ALUSelB=10,ALUOp=add,ExtOp=11,MemtoReg=RegDst=PCWr=PCWrCond=IRWr=MemWr=BrWr=R-type=0,其余任意。

15. 假定图6.6所示的多周期数据通路中寄存器堆只有一个读口和一个写口,若要完成和原数据通路相同的功能,则需要对图中原数据通路做哪些修改?与之对应的有限状态机又如何修改?每条指令的时钟周期数有什么变化?

【分析解答】

如果图6.6所示的多周期数据通路中寄存器堆改成只有一个读口,那么,原来可以同时读数据到A和B,现在只能先读一个到A,再读一个到B。

由于A和B共用一个读地址端口,所以在读地址端口的输入端要加一个多路选择器及其控制信号 RegRead,用于选择读地址口是 rs 还是 rt。可以定义当 RegRead=0 时读口地址为 rs,当 RegRead=1 时读口地址为 rt。

由于只有一个读数据端口,该端口的数据可能被送到A,也可能被送到B,所以读数据端口的输出同时连到A和B的输入端,并增加一个控制信号 AWr,A和B两个寄存器也各自增加一个"写使能"控制端。可以定义当 AWr=1 时将端口数据送A,在 AWr=0 时将端口数据送B。因此,在数据通路中,AWr 信号直接连到A的"写使能"线,而将 AWr 信号取反后连到B的"写使能"线。

由于每个时钟周期只能读取一个操作数,修改后的数据通路不能像原先的数据通路那样,在第2个周期(译码/取数周期)中同时读取 rs 和 rt,必须修改相应的有限状态机。可以有以下3种修改方式。

(1) 将原来的译码/取数周期再分成两个周期,分别读 rs 到A和读 rt 到B,并在其中一个周期中投机完成转移地址计算。这样,每条指令的执行都多了一个时钟周期。

(2) 在原来的译码/取数周期中先读 rs 到A,对于R-型和 Branch 类指令,再增加一个周期来读 rt 到B;对于其他不需用到 rt 内容的指令,则无须增加新的周期。这样,对于每个R-型和 Branch 类指令的执行,都会多一个时钟周期,而其他指令的时钟周期数不变。

(3) 在原来的译码/取数周期中先读 rt 到B,这样,对于R-型指令、Branch 类指令、Load/Store 类指令和 ori 指令等都要再增加一个周期用于读 rs 到A。

显然,上述3种方式中,第(2)种做法得到的综合 CPI 最小。

16. 某高级语言源程序中的一个 while 语句为"while (save[i]==k) i+=1;",若对其编译时,编译器将 *i* 和 *k* 分别分配在寄存器 $s3 和 $s5 中,数组 *save* 的基址存放在 $s6 中,则生成的 MIPS 汇编代码段如下。

```
1    loop:  sll   $t1, $s3, 2      #R[$t1]←R[$s3]<<2,即 R[$t1]=i×4
```

```
2          add   $t1, $t1, $s6     #R[$t1]←R[$t1]+R[$s6],即 R[$t1]=Address of save[i]
3          lw    $t0, 0($t1)       #R[$t0]←M[R[$t1]+0],即 R[$t0]=save[i]
4          bne   $t0, $s5, exit    #if R[$t0]≠ R[$s5] then goto exit
5          addi  $s3, $s3, 1       #R[$s3]←R[$s3]+1,即 i=i+1
6          j     loop              #goto loop
7   exit:
```

假定图 6.4 所示单周期数据通路和图 6.6 所示的多周期数据通路中各主要功能单元的操作时间分别为：存储器 200ps；ALU 和加法器 100ps；寄存器堆（读或写）50ps。在不考虑多路选择器、控制单元、PC、扩展器和线路等延迟的情况下，单周期和多周期处理器的时钟周期至少各为多少？若上述程序段共循环执行 8 次，则在单周期数据通路和多周期数据通路中执行各需要多少纳秒？

【分析解答】

单周期处理器的时钟周期至少为 200＋50＋100＋200＋50＝600ps；多周期处理器的时钟周期至少为 200ps。对于单周期数据通路中的 8 次循环执行，第 1～4 条指令执行了 8 次，第 5～6 条指令执行了 7 次，因此，共用了(4×8＋2×7)×600＝27600ps＝27.6ns。对于多周期数据通路中的 8 次循环执行，sll、add 和 addi 指令都需要 4 个时钟周期，bne 和 j 指令需要 3 个时钟周期，lw 指令需要 5 个时钟周期，因此，一共用了(4＋4＋5＋3)×8×200＋(4＋3)×7×200＝35400ps＝35.4ns。

17. 时钟周期和 CPI 这两个重要参数对处理器性能起着非常关键的作用，因而在处理器设计中需要对这两个参数进行权衡。有些设计者偏向以增大 CPI 为代价换取较高的主频，而另外一些设计者则偏向以较低主频换取 CPI 值的降低。假设在不同指导思想下设计出的两种不同处理器如下：

M1：多周期处理器，数据通路结构类似图 6.6 所示结构，所不同的是，它将写寄存器与存储器读或 ALU 计算合在同一个时钟周期内进行，主频为 3GHz。

M2：多周期处理器，数据通路结构类似 M1，所不同的是，它进一步将有效地址计算和存储器操作合在同一个时钟周期内进行，因而延长了一个节拍内的关键路径，主频降低为 2.5GHz。

已知基准程序 CPUint 2000 中各类指令的频率为 Load 25%，Store 10%，Branch 11%，Jump 2%，ALU 52%。以基准程序 CPUint 2000 为标准，比较两种不同处理器的性能，说明哪个处理器性能更好。找到一种指令序列组合，使得用它来评测时某个处理器性能明显比另一个高。

【分析解答】

参考图 6.6 所对应的有限状态转换图 6.7，对 M1 和 M2 进行分析，得到如下结果：Load、Store、Branch、Jump 和 ALU 五类指令在 M1 中的 CPI 分别为 4、4、3、3、3，在 M2 中的 CPI 分别为 3、3、3、3、3。因此，M1 和 M2 的综合 CPI 分别为：

$$\text{CPI(M1)} = 25\% \times 4 + 10\% \times 4 + 11\% \times 3 + 2\% \times 3 + 52\% \times 3 = 3.35$$

$$\text{CPI(M2)} = 25\% \times 3 + 10\% \times 3 + 11\% \times 3 + 2\% \times 3 + 52\% \times 3 = 3$$

因此，M1 和 M2 的 MIPS 数分别为：

$$\text{MIPS(M1)} = 3\text{G}/3.35 = 895.5$$

$$MIPS(M2) = 2.5G/3 = 833.3$$

M1 和 M2 分别对图 6.6 所示的数据通路做了不同的改变，显然，M1 的做法效果更好。但当所有指令都是 Load/Store 指令时，M2 的 CPI 还是 3，但 M1 的 CPI 变为 4，其 MIPS 数为 3G/4=750。显然，这种情况下 M2 的性能比 M1 高。

18. 对于多周期 MIPS 处理器，若将数据访问过程分成两个时钟周期，则可使时钟频率从 4.8GHz 提高到 5.6GHz，但会增加 lw 和 sw 指令的时钟周期数。已知基准程序 CPUint 2000 中各类指令的频率为 Load 25%，Store 10%，Branch 11%，Jump 2%，ALU 52%。若以基准程序 CPUint 2000 为标准，则时钟频率提高后处理器的性能提高了多少？若将取指令过程再分成两个时钟周期，则可进一步使时钟频率提高到 6.4GHz，此时，时钟频率的提高是否也能带来处理器性能的提高？为什么？

【分析解答】

假设 M1、M2 和 M3 分别表示时钟频率为 4.8GHz、5.6GHz 和 6.4GHz 的多周期处理器，对如图 6.7 所示的有限状态转换图进行分析得知，Load、Store、Branch、Jump 和 ALU 类指令在 M1 中的 CPI 分别为 5、4、3、3、4，在 M2 中的 CPI 分别为 6、5、3、3、4，在 M3 中的 CPI 分别为 7、6、4、4、5。因此，用基准程序 CPUint 2000 来计算，得到它们的综合 CPI 分别为：

$$CPI(M1) = 25\% \times 5 + 10\% \times 4 + 11\% \times 3 + 2\% \times 3 + 52\% \times 4 = 4.12$$

$$CPI(M2) = 25\% \times 6 + 10\% \times 5 + 11\% \times 3 + 2\% \times 3 + 52\% \times 4 = 4.47$$

$$CPI(M3) = 25\% \times 7 + 10\% \times 6 + 11\% \times 4 + 2\% \times 4 + 52\% \times 5 = 5.47$$

因此，M1、M2 和 M3 的 MIPS 数分别为：

$$MIPS(M1) = 4.8G/4.12 = 1165$$

$$MIPS(M2) = 5.6G/4.47 = 1253$$

$$MIPS(M3) = 6.4G/5.47 = 1170$$

由此可见，M2 中将数据访问改为双周期的做法效果较好。M3 中进一步把取指令改为双周期的做法反而使 MIPS 数变小了，因此，这种做法不可取。产生上述结果的原因在于，数据访问只涉及 Load/Store 指令，而取指令则涉及所有指令，使得 CPI 显著增大，从而降低了性能。

19. 假定有一条 MIPS 伪指令“bcmp \$t1, \$t2, \$t3”，其功能是实现对两个主存块数据的比较。\$t1 和 \$t2 中分别存放两个主存块的首地址，\$t3 中存放数据块的长度，每个数据占一个字(4 个字节)。若所有数据都相等，则将 0 置入 \$t1；否则，将第一次出现不相等时的地址分别置入 \$t1 和 \$t2 并结束比较。

(1) 若 \$t4 和 \$t5 是两个空闲寄存器，请给出利用 \$t4 和 \$t5 实现该伪指令的指令序列。

(2) 假定比较的数据块大小为 50 个字，说明在类似于图 6.6 所示的多周期数据通路中执行该伪指令时最多需要多少个时钟周期。

【分析解答】

(1) 实现伪指令“bcmp \$t1, \$t2, \$t3”的指令序列如下。

```
          beq   $t3, $zero, done          #若数据块长度为 0,则结束
compare:  lw    $t4, 0($t1)               #块 1 的当前数据取到$t4
          lw    $t5, 0($t2)               #块 2 的当前数据取到$t5
```

```
        bne   $t4, $t5, done          #$t4和$t5的内容不等,则结束
        addi  $t1, $t1, 4             #块1中的当前数据指向下一个
        addi  $t2, $t2, 4             #块2中的当前数据指向下一个
        addi  $t3, $t3, -1            #比较次数减1
        bne   $t3, $zero, compare     #若没有全部比较完,则继续比较
        addi  $t1, $zero, 0           #若全部都相等,则将$t1置0
done:
```

(2) 在类似图 6.6 所示的多周期数据通路中执行时，上述程序段中用到的指令 beq、lw、bne 和 addi 的时钟周期数分别为 3、5、3 和 4。若比较的数据块大小为 50 个字，则上述指令序列中的循环最多被执行 50 次，因而所需要的指令数最多为 1＋50×7＋1＝352。其中，Load 指令为 50×2＝100 条，时钟周期数为 5×100＝500；Branch 指令数为 1＋2×50＝101，时钟周期数为 3×101＝303；addi 指令数为 1＋3×50＝151，时钟周期数为 4×151＝604。所以，总时钟周期数最多为 500＋303＋604＝1407。

20. 在一个指令集中增添功能强大的复杂指令时，通常使用微程序方式对这些复杂指令进行控制。如果要求在图 6.6 所示的多周期数据通路中采用微程序控制方式实现第 19 题中的"bcmp rs, rt, rd"指令，并且规定使用通用寄存器作为两个额外临时寄存器 rt1 和 rt2。

(1) 请给出全套的设计方案，包括该指令的汇编形式、机器码格式(与原有指令格式兼容)、指令执行流程(用 RTL 表示)、数据通路修改方案、控制信号增加或修改方案、指令执行对应的有限状态机等。

(2) 与第 19 题的软件实现方式相比，该指令用硬件实现时的速度快了多少？硬件实现速度快的原因是什么？

(3) 能否用单周期数据通路实现？假定能，则用软件实现合算还是用硬件实现合算？

(4) 如果 rt1 和 rt2 不用通用寄存器，而是用内部寄存器，则会有哪些好处？

【分析解答】

(1) 该指令的汇编形式为"bcmp rs, rt, rd, rt1, rt2"。

该指令的机器码格式如下。其中 6 位 OP 字段可以选择一个未被其他指令使用的编码。

31　26	25　21	20　16	15　11	10　6	5　1	0
Op	rs	rt	rd	rt1	rt2	0

指令的功能：比较个数由 rd 指出，如果 rd 的内容为 0 则什么都不做，继续执行下一条指令，否则，对 rs 和 rt 所指内存单元的数据依次比较，直到发生以下情况为止。①若存在一对数据不相等，则将不相等的数据所在的内存单元地址分别保存在 rs 和 rt 中，结束执行；②若所有数据都相等，则将 0 存放在 rs 中，结束执行。rt1 和 rt2 是临时寄存器，在指令执行过程中，rs、rt、rd、rt1 和 rt2 都会被破坏。

该指令执行流程如图 6.14 所示。

对图 6.6 所示的数据通路和控制信号做以下修改，以支持该指令的执行。

① 在存储器的 Adr 处对原 MUX 进行修改，增加 R[rs](在 A 中)和 R[rt](在 B 中)两个输入，原控制信号 IorD 改为 2 位，当 IorD 为 0、1、2 和 3 时，分别将 PC、ALUout、A 和 B

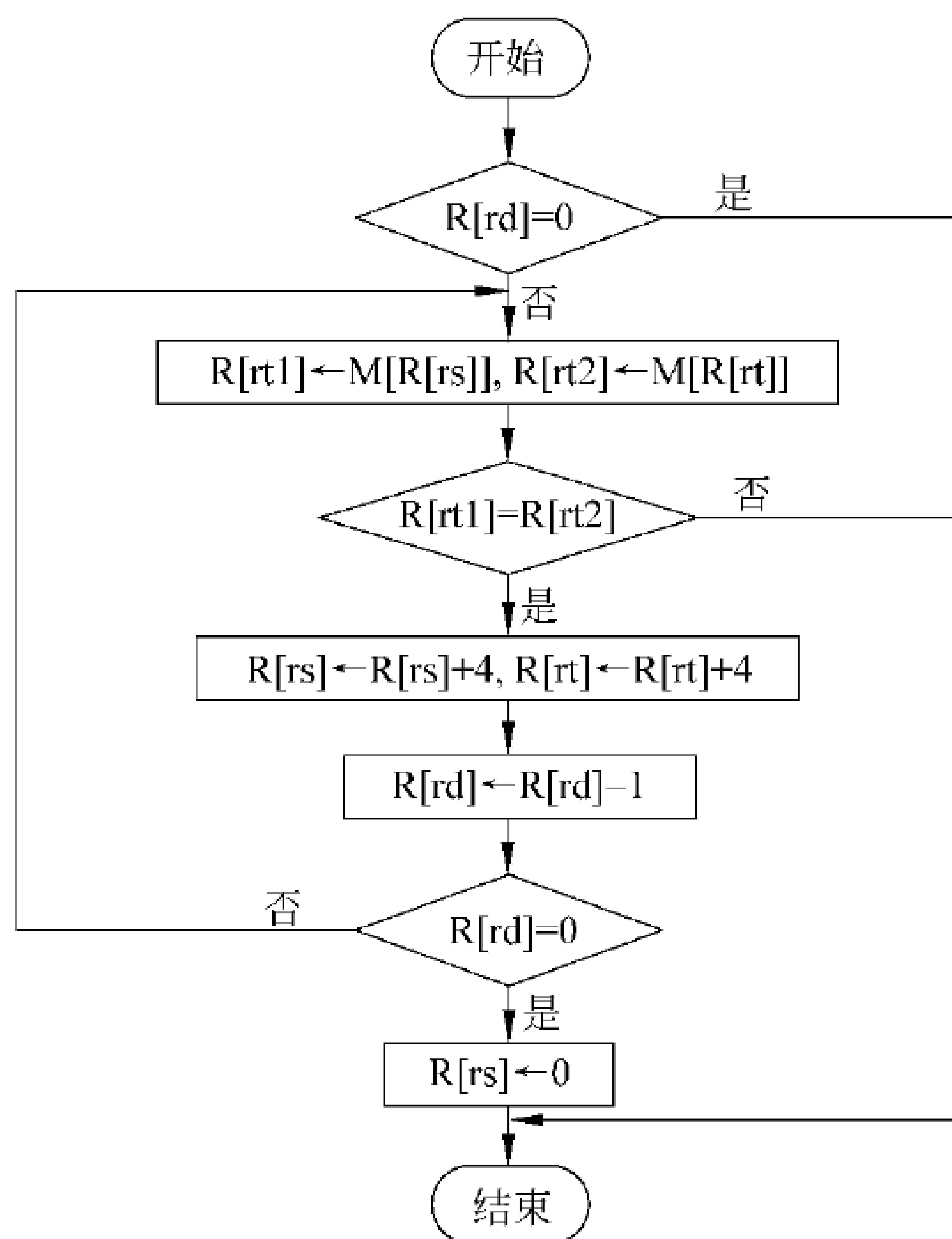


图 6.14　指令执行流程图

送 Adr 处。

② 在寄存器堆的 Ra 处增加一个 MUX，除原来的 rs 外，增加 rt、rd、rt1 三个输入端，并相应地增加 2 位控制信号 RdAdd1。当 RdAdd1 为 0、1、2 和 3 时，寄存器读地址分别为 rs、rt、rd 和 rt1。

③ 在寄存器堆的 Rb 处增加一个 MUX，除原来的 rt 外，另一个输入来自 rt2，同时增加相应的 1 位控制信号 RdAdd2，当 RdAdd2 为 0 和 1 时，寄存器读地址分别为 rt 和 rt2。

④ 在寄存器堆的 Rw 处对原 MUX 进行修改，除了原来的 rt 和 td 外，增加 rs、rt1 和 rt2 三个输入，原控制信号 RegDst 改为 3 位，当 RegDst 为 0、1、2、3、4 时，寄存器写地址分别为 rs、rt、rd、rt1 和 rt2。

⑤ 在寄存器堆的 Dw 处对原 MUX 进行修改，除了原来的输入以外，增加一个输入端"0"，原控制信号 MemtoReg 改为 2 位，当 MemtoReg 为 0、1 和 2 时，存入寄存器的内容分别为 ALU 输出结果、存储器读出数据和 0。

⑥ 在 ALU 的 B 输入端处对原 MUX 进行修改，再增加两个输入"0"和"1"，以使 ALU 能执行"R[rd]－0"和"R[rd]－1"的操作，这样，原控制信号 ALUSelB 改为 3 位，ALUSelB 为 0、1、2、3 时，原先的定义不变，ALUSelB 为 4 和 5 时，ALU 的 B 输入端分别为 0 和 1。

根据该指令执行流程，以及相应数据通路和控制信号的定义，得到该指令执行过程对应的状态转换图，如图 6.15 所示。图中在每个状态内仅给出了关键控制信号的取值，并省略了最初的两个公共状态（取指令周期和译码/取数周期）以及其他指令执行的状态（参见图 6.7）。该指令的执行从第 12 状态开始，每个状态按顺序编号，分别为 12、13、…、25。

图 6.7 对应的原 11 条指令的有限状态机已有 12 个状态和两个分支点。加上该指令的 3 个分支点，新的有限状态机中有 5 个分支点和 26 个状态，如果用微程序设计方式实现该

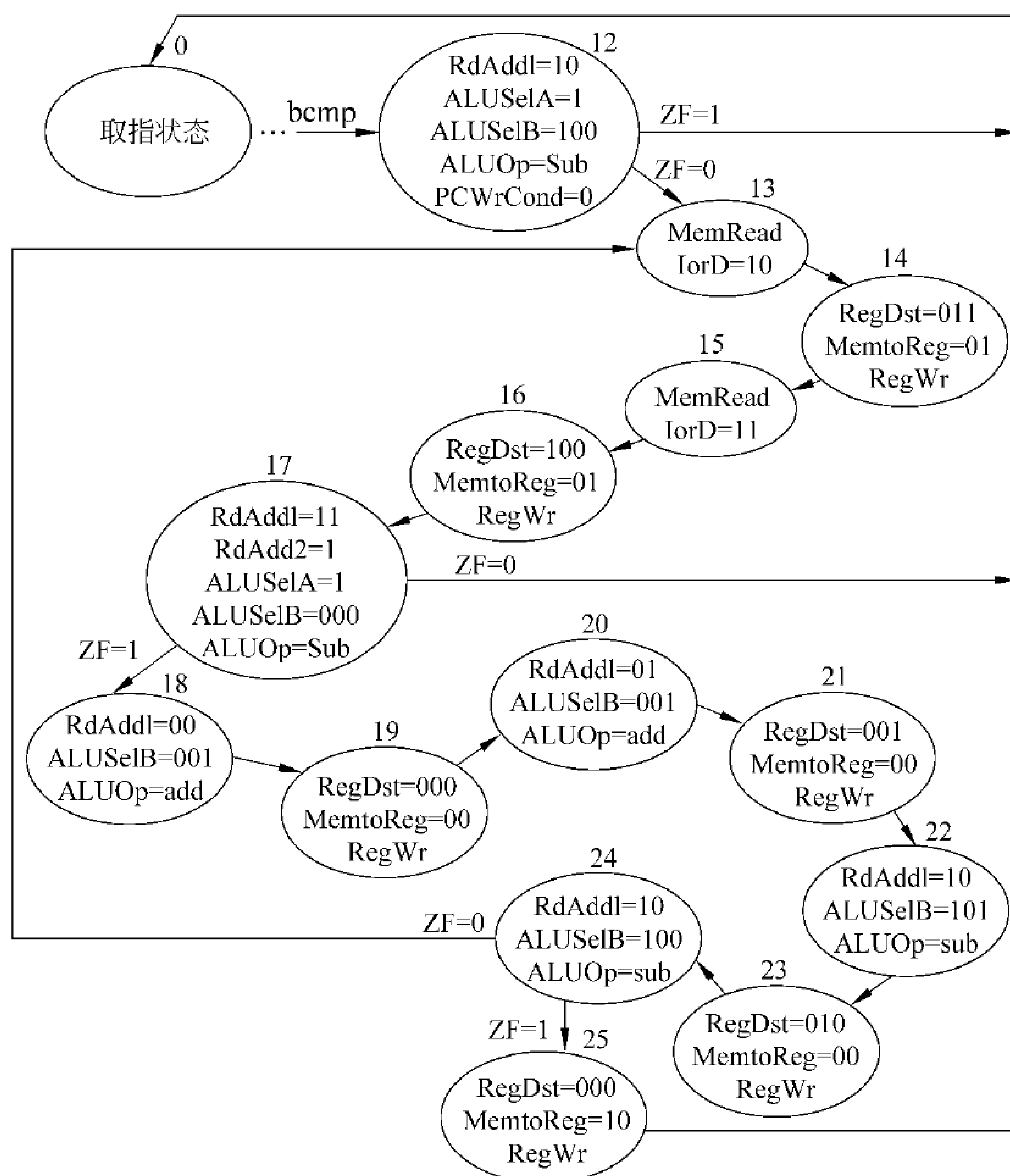


图 6.15 指令执行的有限状态转换图

指令的控制逻辑，每个状态对应一条微指令，那么，微指令地址应该有 5 位，转移控制字段 BrCtr 的位数为 3，其编码及含义如下(Next μaddr 为下一条微指令地址)。

000	Next μaddr=0
001	Next μaddr=ROM1 中对指令译码得到的相应入口地址
010	Next μaddr=ROM2 中 lw/sw 对应的入口地址
011	Next μaddr=μaddr+1
100	Next μaddr=该指令分支 1 处输出的地址
101	Next μaddr=该指令分支 2 处输出的地址
110	Next μaddr=该指令分支 3 处输出的地址

执行到该指令时，则调出该指令对应的微程序执行。该指令对应的微程序是从第 12 状态开始的微指令序列。其中的 3 个分支点分别在第 12、17 和 24 状态，都是根据标志 ZF 控制分支的。对于第 12 状态的分支 1，若 ZF=1，则 Next μaddr=0，否则 Next μaddr=μaddr+1；对于第 17 状态的分支 2，若 ZF=0，则 Next μaddr=0，否则 Next μaddr=μaddr+1；对于第 24 状态的分支 3，若 ZF=0，则 Next μaddr=01101，否则 Next μaddr=μaddr+1。因此，分支 1 和分支 2 处可以各用一个 MUX 实现，根据 ZF 的值选择将 0 还是 μaddr+1 输出作为

Next μaddr 的值。分支 3 处用一个地址修改逻辑，当 ZF＝0 时，将输入 11001 改为 01101，ZF＝1 时不修改。对应的下一条微地址生成逻辑如图 6.16 所示，在此假定采用计数器法。

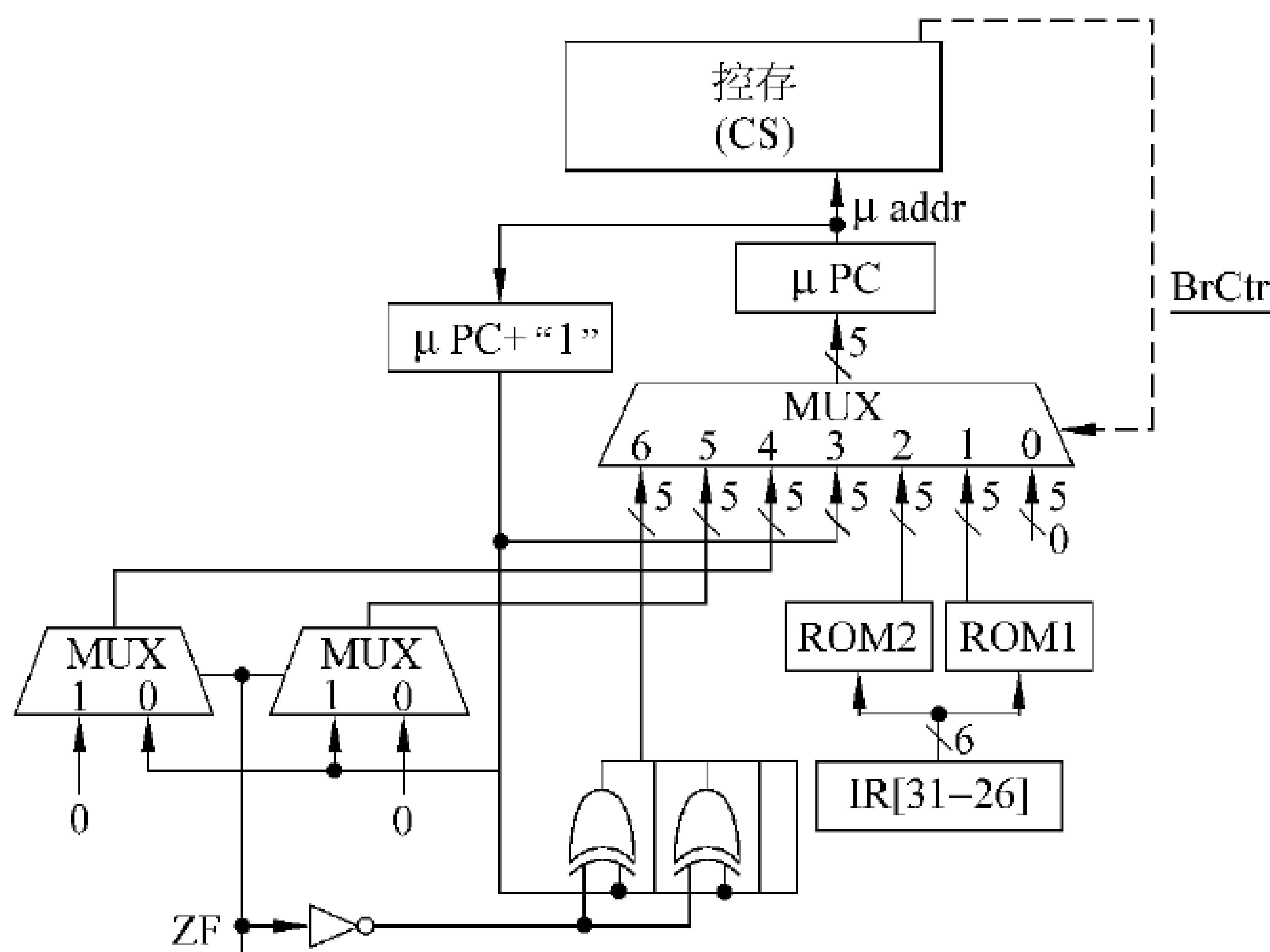


图 6.16　题 20 中的下一条微地址生成逻辑

(2) 假定 bcmp 指令的比较次数为 $n(\neq 0)$，根据上述有限状态机可知，最坏情况下用硬件实现该指令所需要的时钟周期数为 $2+1+12n+1=4+12n$；而根据第 19 题给定的指令序列可知，最坏情况下用软件实现所需要的时钟周期数为 $10n+(3+6n)+(4+12n)=7+28n$。因此，硬件实现比软件实现至少快一倍多。硬件实现比软件快的原因主要有两个：①软件方式下，指令序列中的每条指令都要取指令、译码/取数，而硬件实现时不需要。②软件方式下，每条指令都要保存结果，下一条指令用时再取，而硬件方式下，中间数据可以直接使用，无须先存后取。

(3) 因为本指令在执行过程中需要多次写寄存器，因此，不能用单周期数据通路实现。假设能用，因为单周期数据通路的时钟周期以最复杂指令为准，因此，这个复杂指令会大大延长时钟周期，此时，用软件实现更合算。

(4) 如果 rt1 和 rt2 不用在指令中明显给出，即不用通用寄存器而改用内部寄存器，则可大大简化数据通路、减少控制信号和时钟周期数，并且可减少被该指令破坏的通用寄存器的个数。

21. 假设微程序控制器容量为 1024×48 位，微程序可在整个控存内实现转移，反映所有指令执行状态转换的有限状态机中有 4 个分支点，采用水平型微指令格式，并采用断定法确定下一条微地址，即由专门的下地址字段确定微地址。请设计微指令的格式，说明各字段的含义和位数，并对转移控制字段进行编码。

【分析解答】

微程序控制器容量为 1024×48 位，说明微地址 10 位，微指令字 48 位。微指令字分 3 个字段：微命令字段、转移控制字段和下地址字段。因为微地址占 10 位，所以下地址字段有 10 位，用来给出下条微指令的地址；因为需要对 5 种情况进行控制，所以转移控制字段有 3 位；剩下的 $48-10-3=35$ 位是微操作码(微命令)字段，用于对微命令进行编码。

采用断定法时，转移控制字段可按以下方案进行编码。

000：下地址字段指出的地址作为下一条微地址

100：根据分支 1 处的条件来选择下一条微地址

101：根据分支 2 处的条件来选择下一条微地址

110：根据分支 3 处的条件来选择下一条微地址

111：根据分支 4 处的条件来选择下一条微地址

如果采用计数器法，则转移控制字段需要对 6 种情况进行控制，在上述 5 种情况的基础上再加上“顺序执行下一条微指令”的情况。其中第一种情况改为“取指微程序首地址作为下一条微地址”。此时，转移控制字段也只要 3 位编码。

22. 对于多周期 CPU 的异常和中断处理，回答以下问题：

(1) 对于除数为 0、溢出、无效指令操作码、无效指令地址、无效数据地址、缺页、访问越权和外部中断，CPU 在哪些指令的哪个时钟周期能分别检测到这些异常或中断？

(2) 在检测到某个异常或中断后，CPU 通常要完成哪些工作？简要说明 CPU 如何完成这些工作。

(3) TLB 缺失和 cache 缺失各在哪个指令的哪个时钟周期被检测到？如果检测到发生了 TLB 缺失和 cache 缺失，那么，CPU 各要完成哪些工作？简要说明 CPU 如何完成这些工作。(提示：TLB 缺失可以有软件和硬件两种处理方式)

【分析解答】

(1) 多周期 CPU 中不同指令执行时可能会发生不同的异常事件，这些异常事件发生在不同的时间，CPU 在不同的时钟周期检测不同的异常事件。

① “除数为 0”异常在取数/译码周期进行检测。

② “溢出”异常在 R-型指令和 I-型运算类指令的执行周期进行检测。

③ “无效指令”异常在取数/译码周期进行检测。

④ “无效指令地址”、“缺页”和“访问越权”异常在取指令周期检测。

⑤ “无效数据地址”、“缺页”和“访问越权”异常在存储器访问周期检测。

⑥ “外部中断”可在每条指令的最后一个周期进行检测。

(2) CPU 检测到某个异常或中断后，要完成的工作是关中断、保护断点和程序状态、识别异常事件(或中断源)并转异常(或中断)处理。

CPU 通过将中断允许触发器清 0 来关中断；计算断点值并将断点和程序状态字寄存器信息送到堆栈或特定的寄存器中；异常的原因和外部中断源可以采用软件识别或硬件识别方式进行识别。

(3) TLB 缺失和 cache 缺失都是在存储访问过程中发生的，因而这两种缺失在相同的周期内进行检测。与存储访问相关的周期有每个指令的取指令(IF)周期、lw 指令和 sw 指令的存储器访问(Mem)周期。对于 TLB 缺失，可以有硬件和软件两种处理方式，如果是硬件处理 TLB 缺失，则在 CPU 中有专门处理 TLB 缺失的逻辑部件，能够启动一次存储器读操作，到主存读取相应的页表项内容装入 TLB；如果是软件处理 TLB，则在 CPU 检测到发生了 TLB 缺失的情况下，就发出“TLB 缺失”异常，调出专门的异常处理程序进行处理。对于 cache 缺失，则通常采用硬件处理方式。CPU 中必须提供 cache 缺失处理逻辑，当 CPU 检测到 cache 缺失时，由硬件自动启动存储器读操作，读取一个“主存块”到 cache 中。

第 7 章 指令流水线

7.1 教学目标和内容安排

主要教学目标：

使学生深刻理解现代计算机的CPU是如何执行指令的，包括指令流水线基本原理、流水段寄存器的概念、流水线数据通路的设计、流水线的控制信号、结构冒险及其处理、数据冒险及其处理、转发技术、控制冒险及其处理、分支预测原理、超标量和动态流水线的概念；并让学生初步了解在指令流水线中如何处理异常和中断，以及各种存储器访问缺失对指令流水线的影响。

基本学习要求：

(1) 理解指令流水线的一般概念。

(2) 了解适合流水线执行的指令集特征。

(3) 掌握流水线数据通路的设计方法。

(4) 了解流水线控制器的设计原理。

(5) 理解流水线冒险的基本概念。

(6) 了解结构冒险的概念和处理策略。

(7) 了解数据冒险(数据相关)的概念。

(8) 了解运用转发技术解决数据冒险的基本原理。

(9) 了解"Load-use"数据冒险的概念和处理策略。

(10) 了解控制冒险的概念和引起控制冒险的几种原因。

(11) 了解静态分支预测和动态分支预测的基本原理。

(12) 了解异常和中断对流水线的影响。

(13) 了解访存缺失对流水线的影响。

(14) 了解超流水线、超标量和动态流水线等高级流水线的基本概念及其基本实现技术。

现代计算机都采用流水线方式执行指令，因此，有关指令的流水线执行方式和流水线处理器的实现等内容是非常重要的。主教材[①]依照"最早的简单累加器型CPU→总线式CPU→单

① 主教材指《计算机组成与系统结构》(袁春风编著，清华大学出版社，2010.4)

周期 CPU→多周期 CPU→基本流水线 CPU→动态超标量超流水线 CPU”的次序，循序渐进地介绍了 CPU 设计技术及其发展过程。实际上前面介绍的总线式 CPU、单周期 CPU 和多周期 CPU 等非流水线 CPU 设计的内容都是为后面介绍流水线 CPU 设计做铺垫的，因为，真正需要学生掌握的应该是反映现实的流水线处理器的实现技术。

主教材主要从流水线概述、流水线处理器的实现、流水线冒险及其处理和高级流水线技术 4 个方面对流水线技术进行了介绍。

对于流水线概述部分，因为其内容比较简单，所以，只要通过对一个简单例子的讲解，让学生明白采用流水线方式执行指令的基本思想，以及采用流水线后指令的吞吐率与指令执行时间的变化。

对于流水线处理器的实现，以具有代表性的 11 条 MIPS 指令为实现目标，分析每条指令的功能和执行过程，找出最复杂指令所需要的执行阶段。在介绍流水线数据通路设计时，可以对照前面的单周期数据通路设计思路，对两者进行比较，以使学生有一个从简单到复杂的认识过程，有利于学生对流水线设计和实现的基本方法的掌握。在课时有限的情况下，可以只选择对部分指令在流水线中的执行过程进行讲述。对于流水线 CPU 中控制器的设计，只要把控制信号在流水段寄存器中的传送过程说明清楚，再说明流水线 CPU 中控制器设计方式与单周期 CPU 中控制器设计方式是一样的就可以了。

对于流水线冒险及其处理，可以通过具体指令序列在流水线数据通路中的执行过程来讲解结构冒险、数据冒险和控制冒险 3 种基本流水线冒险的含义和相应的冒险处理原理(例如，加 nop 指令、转发、阻塞、分支预测等)。课时有限的情况下，对于一些具体的实现细节可以跳过，如具体的转发线路、带转发控制的流水线数据通路、Load-use 数据冒险的检测和处理、分支预测的具体实现、异常或中断引起的控制冒险的具体处理、访问缺失引起的流水线阻塞处理等。虽然，上述内容的具体实现在课时不够时可以不讲，但是，对于这些内容所涉及的基本概念和基本原理，学生还是应该有一定程度的了解，因为这些内容对学生全面了解指令执行过程中数据的流动过程，以及全面掌握现代计算机处理器设计技术是必需的。

对于高级流水线技术，基本要求是能够了解超流水线、超标量和动态流水线等高级流水线的基本概念及其基本实现技术。在课时有限的情况下，课堂中只要简单说明每种实现技术的基本思想即可，具体的实现实例和详细的描述部分可以跳过。

7.2 主要内容提要

1. 指令流水线的设计和实现

指令流水线设计的基本思想是，将每条指令的执行规整化为同样的几个流水阶段，并规定每个流水阶段的执行时间一样，都等于一个时钟周期。采用流水线方式执行指令，其吞吐率比非流水线方式下提高了若干倍，但是，对于每一条指令来说，反而比非流水线方式时延长了执行时间。

每个流水段中的部件都是一组组合逻辑加上一组寄存器，组合逻辑中产生的结果在时钟到来时被存到寄存器(如程序计数器、条件码寄存器、流水段寄存器)中。每两个相邻流水段之间的流水段寄存器，用于记录所有在后面阶段要用到的各种信息，例如，指令代码、参加运算的操作数、指令运算结果、指令异常信息、寄存器读口地址、寄存器写口地址、存储单元

地址、新的PC值等。指令译码得到的控制信号也通过流水段寄存器传送到后面各个流水段中。

2. 指令流水线的局限性

在理想情况下，每个时钟到来，都有一条指令进入流水线，也有一条指令执行结束。但是，很多因素会导致指令流水线的情况不很理想。首先，并不是每条指令都有相同多个流水段，也不是每个流水段的执行都需要一样长的时间，因此，为了能够方便地控制指令流水线的执行，通常以最复杂指令所需阶段数来确定流水段个数，并以最复杂阶段所需时间为基准来设计时钟周期；其次，随着流水线深度的增加，流水段寄存器的读写所带来的额外开销比例也增大；最后，指令执行时，还会发生资源冲突、数据相关、控制相关、cache缺失等问题，导致流水线被阻塞而延长程序执行时间。

3. 流水线数据通路设计举例

以下以MIPS处理器为例，概要说明流水线数据通路的实现原理。与第6章中单周期数据通路和多周期数据通路的实现目标一样，流水线数据通路的实现目标也是表6.1所示的11条MIPS指令。根据对11条指令的分析，可以得到执行这11条指令的五段流水线数据通路基本框架，如图7.1所示。

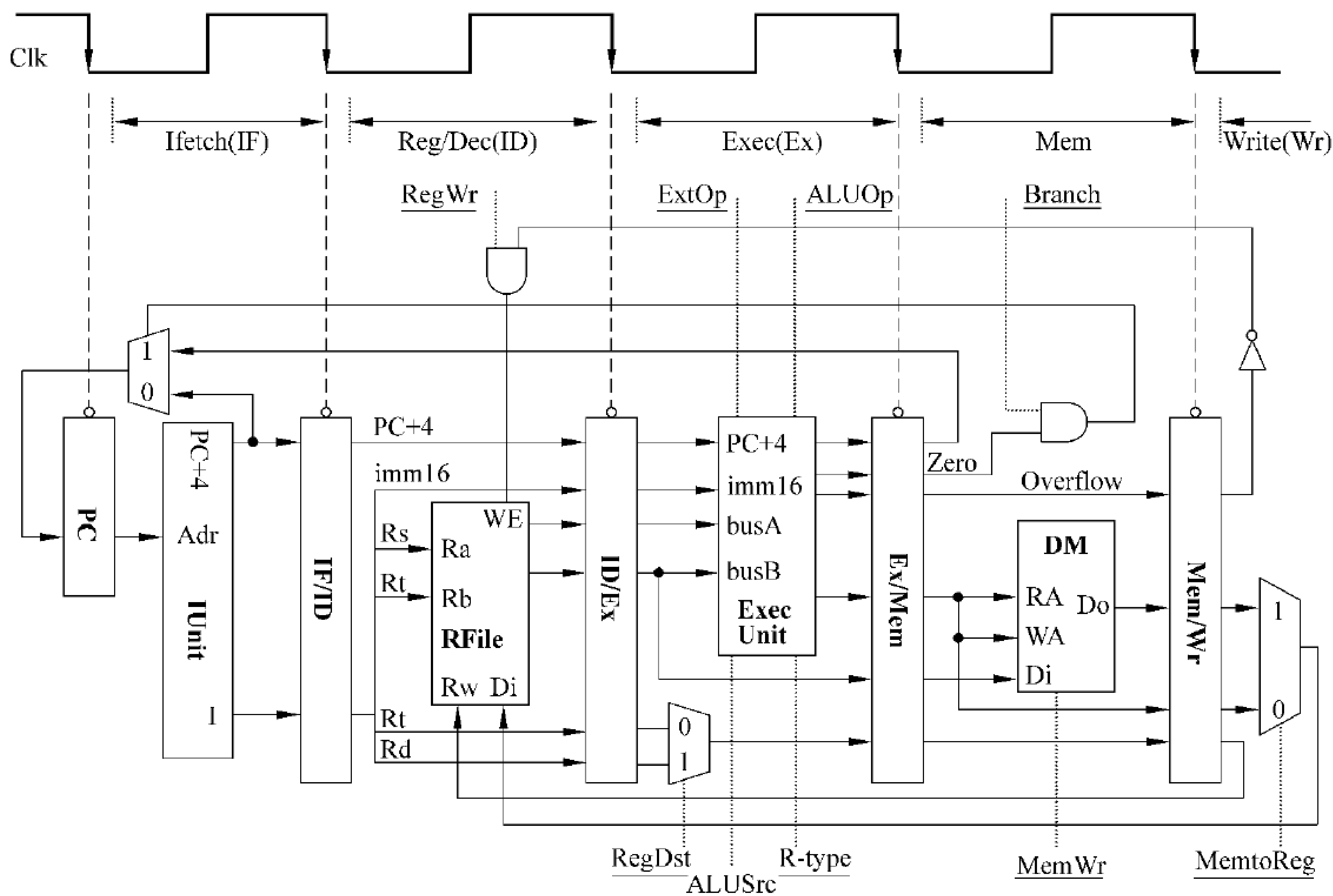


图7.1 五段流水线数据通路基本框架

在图7.1所示的流水线数据通路中，每条指令的执行都经历5个流水段：IF、ID、Ex、Mem和Wr，每个流水段都在不同的功能部件中执行。流水段之间有一个流水段寄存器，例如，IF/ID寄存器是介于IF段和ID段之间的寄存器。每个流水段寄存器用来存放从当前流水段传到后面所有流水段的信息。因为每个段间传递的信息不一样，所以各流水段寄存器的长度也不一样。

图 7.1 中的取指令部件 IUnit 和执行部件 ExecUnit 分别如图 7.2 和 7.3 所示。

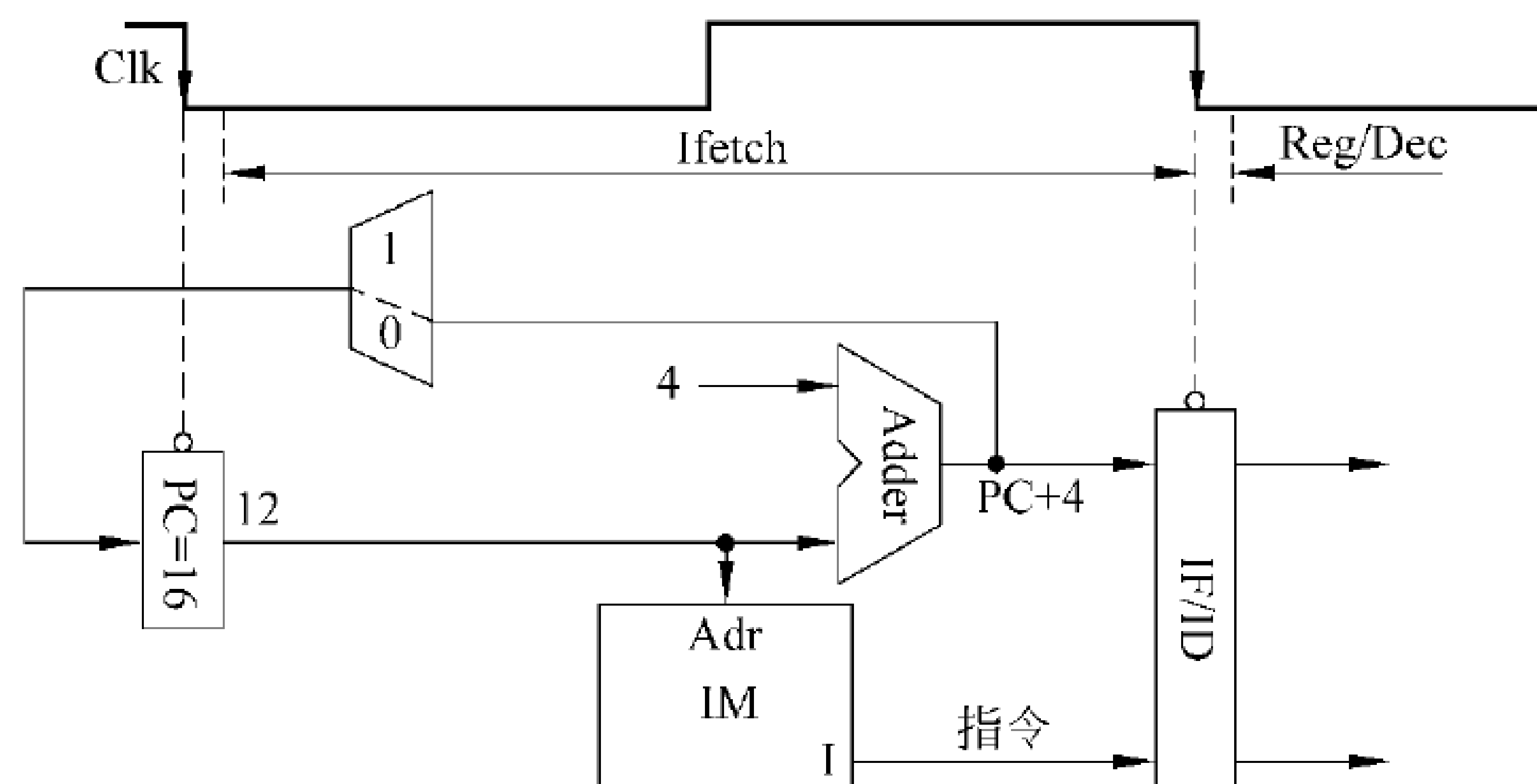


图 7.2　取指令部件 IUnit 的实现

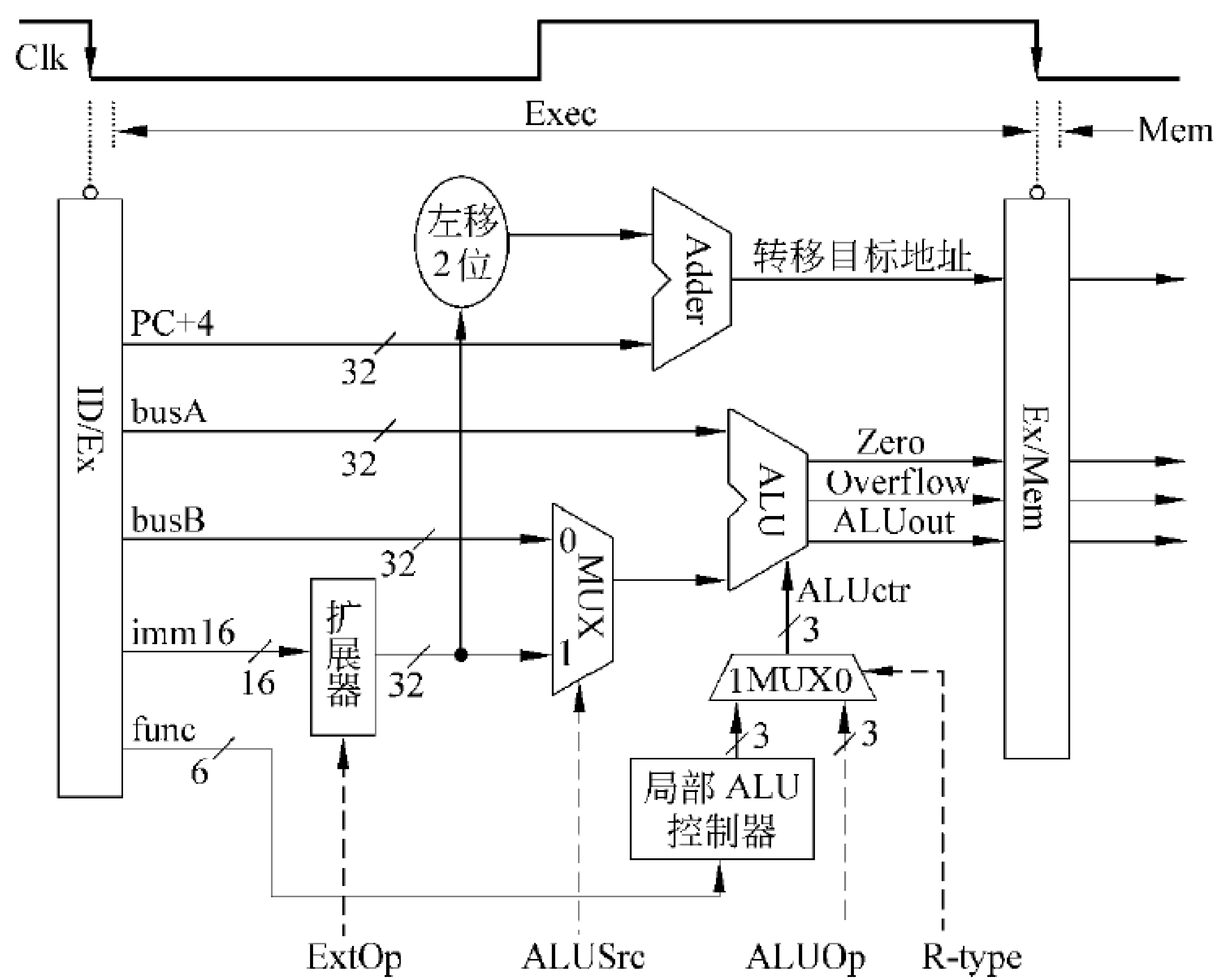


图 7.3　执行部件 ExecUnit 的实现

如图 7.2 所示的取指令部件中，IM 为指令存储器，取出的指令被送到 IF/ID 流水段寄存器的输入端，在下一个时钟到来后开始写入。在取指令的同时，计算 PC＋4，得到的值被送到 PC 的输入端，在下一个时钟到来后开始写入 PC。

不同的指令在不同的控制信号控制下在执行部件中执行不同的操作。

4. 流水线冒险的检测与处理

指令流水线中，可能会遇到一些情况使得流水线无法正确执行后续指令，而引起流水线阻塞（停顿）。这种现象称为流水线冒险。根据导致冒险的原因的不同，分为结构冒险、数据冒险和控制冒险 3 种。

结构冒险也称资源冲突，由多条指令同时竞用同一个功能部件引起。所用的解决策略包括：①规定每个功能部件在一条指令中只能被用一次；②规定每个功能部件只能在某个特定的阶段被用；③将指令存储器（如 code cache）和数据存储器（如 data cache）分开。

数据冒险也称数据相关。当前面指令的执行结果是后面指令的操作数时，可能会发生数据冒险。所用解决策略有软件和硬件两种方式。①软件方式：编译器在发生数据相关的指令之间插入足够的 nop 指令，这种方式下，CPU 执行程序时便不会发生数据冒险现象，软件方式简化了硬件实现，但是，软件方式会同时增加程序的空间开销和时间开销；②硬件方式：有两种硬件解决方式，一种方式是单纯采用“阻塞”技术，另一种方式是采用“转发＋阻塞”技术。在上述图 7.1 中加上转发检测及处理、Load-use 冒险检测及处理机制后，就可以解决数据冒险问题。采用“转发＋阻塞”技术的部分流水线数据通路如图 7.4 所示。

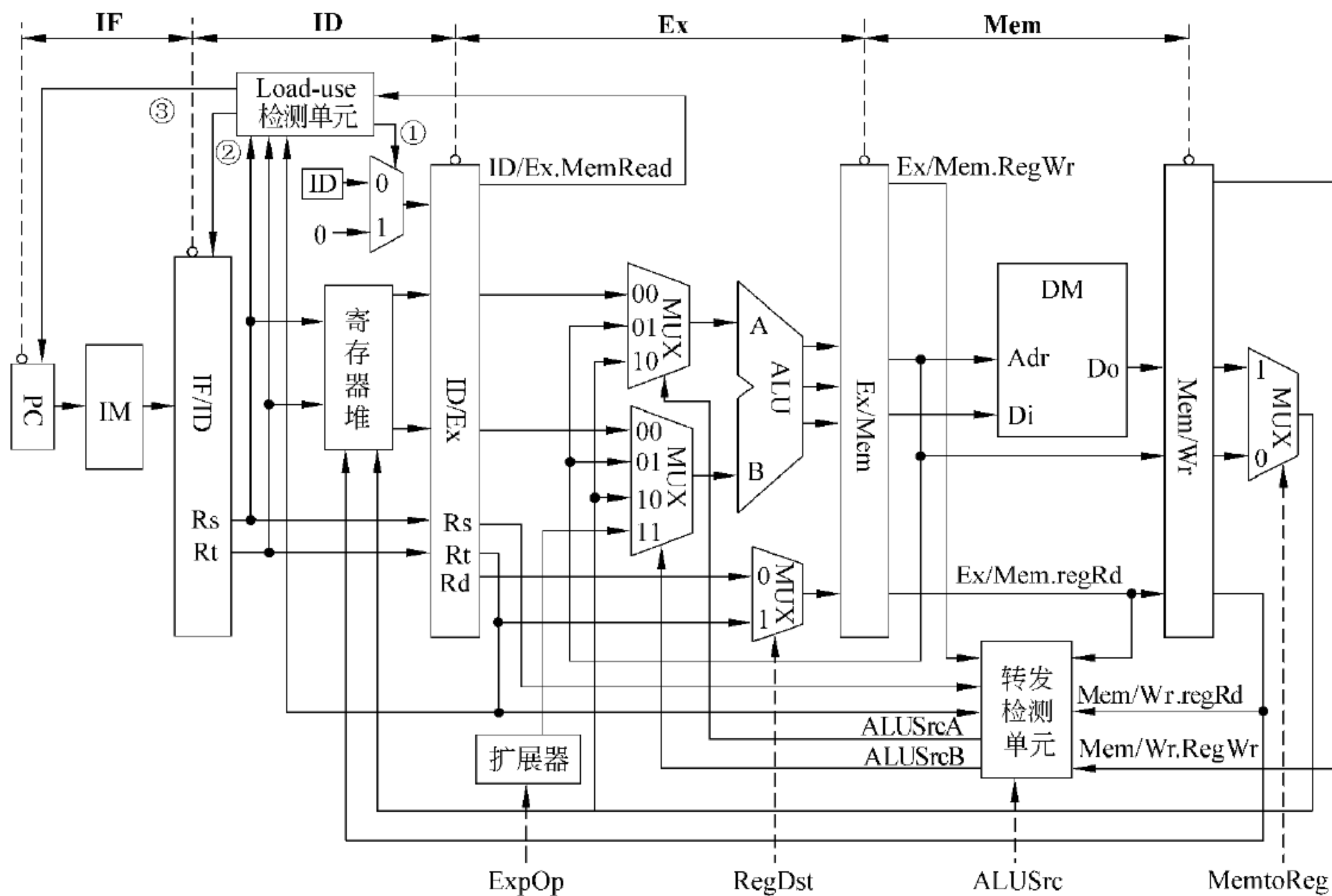


图 7.4 带转发和 load-use 冒险处理的部分流水线数据通路

CPU 在执行程序过程中，动态检测是否存在数据相关，在存在数据相关的情况下，通过转发(旁路)技术将前面指令执行过程中得到的结果直接传送到后面指令执行时需要操作数的地方，对于像 Load-use 这样不能通过转发解决的数据冒险，则在指令的特定流水段插入“气泡”以“阻塞”指令继续执行，直到后面指令能够取得所需数据为止。单纯采用硬件“阻塞”的方式，使得程序的空间开销比软件方式好，但时间开销并没有减少，而采用“转发＋阻塞”的方式则可同时减少空间开销和时间开销。

控制冒险也称控制相关。当返回指令、分支指令、跳转指令等有可能改变顺序增量的 PC 值时，由于获取转移目标地址的时间较长，使得在目标地址产生前已经将下一条指令取到流水线中，如果已经取出执行的指令不是应该执行的指令，则发生了控制冒险。所用解决策略也分为软件方式和硬件方式。①软件方式。有两种软件方式，一种方式是编译器简单地在分支指令、跳转指令等引起控制相关的指令后面插入足够的 nop 指令；另一种方式是分支延迟调度技术，编译器将前面若干条与分支指令等无关的指令调到后面填充到延迟槽中，不够填满延迟槽时，用 nop 指令补足。不同指令的延迟槽(延迟损失时间片)不同，它与

该指令能在几个周期内得到正确的PC值有关。在软件进行了这些处理后，流水线执行时就不会发生控制冒险现象。②硬件方式。在分支指令、跳转指令等引起控制相关的指令后面插入"气泡"，使流水线停顿若干时钟，直到得到正确的PC值为止。

对于分支指令引起的控制冒险，若用硬件方式处理，则通常在上述硬件处理的基础上，还会采用"分支预测"技术。有静态预测和动态预测两种技术，静态预测与分支指令执行的历史情况无关，可以预测总是条件满足(Taken)而转移，也可以预测总是条件不满足(Not Taken)而顺序执行；动态预测利用分支指令执行的历史情况来进行预测，并根据实际执行情况动态调整。可采用一位预测、两位预测，甚至三位预测技术，动态预测能达到90%的成功率。不管采用哪种分支预测技术，只要预测失败，都必须将错误执行的指令从流水线中冲刷掉。

除了上述几种流水线冒险之外，异常和中断的发生、cache缺失、TLB缺失等也都会引起流水线阻塞。对于异常和中断引起的冒险，其处理方式与分支预测错误时的处理类似。对于cache缺失引起的冒险，则无须冲刷指令，只要冻结机器状态若干个时钟周期即可。对于TLB缺失，则可以像缺页一样作为一种内部异常来处理，也可以像cache缺失一样完全由硬件处理。

5. 高级流水线技术

高级流水线技术充分利用指令级并行来提高流水线执行的性能。有两种增加指令级并行的策略。一种是超流水线技术，它通过增加流水线的级数来使更多的指令同时在流水线中重叠执行；另一种是多发射流水线技术，通过同时启动多条指令独立运行来提高指令并行性。

采用多发射技术的处理器称为超标量处理器。要实现多发射流水线必须完成对指令进行打包和冒险处理两个任务，指令打包是指将能并行处理的多条指令同时发射到发射槽中。根据指令打包任务是由编译器静态完成还是由处理器动态完成，可将多发射技术分为静态多发射和动态多发射两类。

静态多发射处理器通过编译器静态推测来完成指令打包和冒险处理任务，指令打包的结果可看成将同时发射的多条指令合并到一个长指令中，因此，也称为超长指令字(VLIW)。动态多发射处理器在指令执行时由处理器硬件进行动态流水线调度来完成指令打包和冒险处理任务，即处理器可以动态地将后面的一些无关指令调到前面先执行。动态流水线调度可以实现"按序发射，按序完成"、"按序发射，无序完成"和"无序发射，无序完成"指令执行模式。

7.3 基本术语解释

指令流水线(Instruction Pipelining)

多条指令重叠执行的一种指令执行方式。流水线方式下，一条指令的执行过程被分成若干个操作子过程(也称为"流水段")，每个子过程由一个独立的功能部件来完成。在同一条流水线中，每条指令所包含的操作子过程个数必须一样，每个子过程所花的时间也要设计成相同的。因此，一般按最复杂的指令来设计流水段个数，以最复杂的子过程来设计流水段的宽度。这样，所有功能部件可以同时执行不同指令的不同子过程中的操作，即第i+1条指令的第k段和第i条指令的第k+1段同时执行。理想情况下，经过若干周期后，流水线

能在每个周期内执行完一条指令。

现代计算机一般把复杂度相近的指令用同一条流水线完成，而把复杂度相差很大的指令安排在不同的流水线中。

流水线深度(Pipeline Depth)

流水段的个数称为流水线深度或流水线级数。

指令吞吐量(Instruction Throughput)

单位时间内处理器执行指令的条数。采用流水线指令执行方式能提高指令的吞吐率，但不能缩短每条指令的执行时间。

流水段寄存器(Pipeline Register)

每两个相邻的流水段之间需要设置一个流水段寄存器，用来存放前一个流水段中产生的并需传输到其后所有流水段的信息，包括各种数据(PC、指令、立即数、运算结果、寄存器号等)和控制信号两大类信息。每个流水段的功能不一样，所需传递的信息也不同，各流水段寄存器的长度也因此而不同。

流水线冒险(Hazard)

当若干指令都已进入流水线开始执行后，如果其中某些后续指令的某些流水段任务不能按时开始执行(若执行就会发生错误)，则说明流水线被破坏，这种现象称为流水线冒险。有3种流水线冒险：结构冒险、控制冒险、数据冒险。

结构冒险(Structural Hazard)

在指令流水线中，同一个部件同时需被不同指令所使用的现象称为结构冒险，也称为资源冲突(Resource Conflicts)。

控制冒险(Control Hazard)

在指令流水线中，转移指令或异常等情况改变了程序执行的流程，而使得在目标地址产生前已被取到流水线中的指令无效的现象称为控制冒险，也称为控制相关(Control Dependency)。分支指令引起的控制冒险称为分支冒险(Branch Hazard)。

数据冒险(Data Hazard)

在指令流水线中，后面指令需要用到前面指令的结果时，前面指令的结果还没产生的现象称为数据冒险，也称为数据相关(Data Dependency)。

流水线阻塞(Pipeline Stall)/**气泡**(Bubble)

流水线中下一条指令不能执行时，就在硬件上加入额外的电路来使得下一条指令延迟若干周期再执行，这种方式称为流水线阻塞或流水线停顿。有时也会形象地称这种做法为在流水线中插入气泡(Bubble)。

空操作(Nop)

空操作就是不做任何动作，而只是在时间上延迟一段时间。有两种情况：(1)为了规整流水线，在某些指令的执行过程中加入空流水段，这种空流水段中的操作称为空操作；(2)为了避免流水线冒险，在相关指令的前后加入 nop 指令，使得流水线停顿若干时钟，等到需要的信息得到后再继续执行后面的指令。

转发(Forwarding)

当后面指令要用到前面指令的结果数据时，前面流水段部件中得到的数据直接通过连线传送到后面流水段的部件中，而不等待前面指令的结束。这种方式称为转发或旁路，它能

解决部分数据冒险。

旁路(Bypassing)

是数据转发技术的别称。

分支条件满足(Branch Taken)/**分支条件不满足**(Branch Not Taken)

对于条件转移指令(分支指令 Branch),其执行结果总有两种可能性:一种是条件满足(称为“Branch Taken”),此时转到转移目的地址处继续执行;另一种是条件不满足(称为“Branch Not Taken”),此时则继续取下一条指令执行。

分支预测(Branch Predict)

在流水线执行过程中,对每条分支指令进行预测,可以简单地预测“Branch Taken”或“Branch Not Taken”,也可以根据历史情况动态预测。流水线执行时,总是把预测发生的分支处下一条指令取到流水线中。这样,只要预测成功,流水线就不会发生阻塞。如果预测不成功,则流水线要退回到正确的分支地址处重新启动流水线执行,原先不该取出执行的指令要排除出流水线,称为清洗或冲刷(Flush)流水线。

分支延迟(Delayed Branch)

分支延迟调度方法采用编译优化方法调整指令执行顺序,将分支指令前面的、与分支指令无关的指令放到分支指令后面执行,以填充延迟损失时间片(分支延迟槽),不够时再用 nop 操作填充,这样,使得分支指令能在足够长的时间内得到跳转地址计算结果,从而避免流水线的阻塞。

分支延迟损失时间片(Penalty for Branch Delay)

由于分支指令引起流水线阻塞而带来的延迟执行时钟周期数,也称为分支延迟时间片。

分支延迟槽(Branch Delay Slot)

分支延迟调度方法中,分支指令后面被填的指令位置称为“分支延迟槽”。需要填入的指令条数(分支延迟槽数)等于分支延迟损失时间片。

指令级并行(Instruction Level Parallelism,ILP)

在 CPU 执行程序时,通过多条指令之间的并行执行来提高处理器性能的一种技术。

IPC(Instruction per Clock)

每个时钟周期内可以执行完成的指令条数,是 CPI 的倒数。

超流水线(Superpipelining)

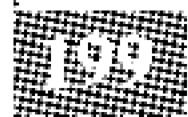
是指具有更多级流水段的一种流水线。在理想情况下,流水线阶段越多,则指令的吞吐率越高,所以一些处理器采用 8 个或更多流水阶段,称为超流水线。

超长指令字(Very Long Instruction Word,VLIW)

通过编译器静态推测来辅助完成指令打包和冒险处理时,通常将一个周期内发射的多个指令预先组织为一条具有多个操作的长指令,这种将多条指令打包生成的指令称为超长指令字,执行这种超长指令字的处理器称为 VLIW 处理器。

超标量流水线(Superscalar Pipelining)

若干条指令(如整数运算、浮点运算、装入/存储等)被同时启动并独立进入流水线执行,即每个时钟周期发射多条指令。为此,需要有多套取指部件和指令译码部件,并且同时有多条指令执行,所以应有多个执行部件。如定点处理部件、浮点处理部件、乘/除法部件、取数/存数部件等。超标量流水线采用的是一种多指令发射(Multiple-instruction issue)方式。



动态流水线(Dynamic Pipelining)

动态流水线通过指令相关性检测和动态分支预测等手段,投机性地不按指令顺序执行,当发生流水线阻塞时,可以到后面找指令来执行。动态流水线的通用模型由以下主要单元组成:指令预取和分发单元、执行单元、提交单元。这种调整指令执行顺序的方式称为动态流水线调度(Dynamic Pipeline Scheduling)。

指令预取(Instruction Prefetch)

在指令执行前,预先把指令取到 CPU 的指令缓冲器中。这样,在指令执行时,不需要再去取指令,可以很快地直接执行指令。流水线指令执行方式中通常采用指令预取。

指令分发(Instruction Dispatch)

对指令队列中的指令进行译码,根据译码结果将指令分派到相应的执行单元。用来进行指令译码和分发的部件称为分发单元。

静态多发射(Static Multiple Issue)

通过编译器静态推测来辅助完成指令打包和冒险处理。通常将一个周期内同时发射的多个指令预先组织为一条多个操作的长指令,称为一个"发射包"。静态多发射处理器主要考虑处理数据冒险和控制冒险。

动态多发射(Dynamic Multiple Issue)

通过在指令执行过程中由处理器硬件动态完成流水线调度来完成指令打包和冒险处理。目前的超标量处理器大多采用动态多发射流水线。

按序发射(In-order Issue)

按照程序中原来的指令顺序进行指令发射。

无序发射(Out-of-order Issue)

不按程序中原来的顺序发射指令,把可能发生冒险的指令推后发射,而把后面无冒险的指令提前发射。

指令执行单元(Instruction Execute Unit)

用来执行指令的功能部件。在动态流水线中,不同的指令被分派到不同的指令执行单元中。如整数运算单元、浮点运算单元、存数单元、取数单元等。

存储站(Reservation Station)

在动态调度流水线中,每个执行单元都有自己的缓存,用来保存当前执行的指令、操作数和结果等。这个缓存称为存储站或保留站。

乱序执行(Out-of-order Execution)

动态多发射流水线中,在指令执行单元有很多指令同时被执行,而且每种指令所用的执行时间也不一样,所以,无法按照程序中原来的顺序执行指令,因此,动态流水线中,指令一定是乱序执行的。

重排序缓冲(Reorder Buffer)

因为指令执行是无序的,所以,需要对执行后的指令重新排序,在指令最后提交前先要将其存放在重排序缓存中。

指令提交单元(Instruction Commit Unit)

将指令执行的结果写到结果寄存器或存储单元中以完成指令最后一步操作称为结果提

交。提交单元在重排序缓冲中保存所有挂起的指令，根据分支功能单元执行的结果确定预测是否成功，从而确定哪些指令需从重排序缓冲中清除，哪些指令可以提交结果。

按序完成(In-order Completion)

按照程序中原来的指令顺序完成指令的执行。

无序完成(Out-of-order Completion)

不按照程序中原来的指令顺序完成指令的执行。

按序发射按序完成

按照程序中原来的指令顺序预取、译码和分发，也按原来的指令顺序完成指令的执行。

按序发射无序完成

按照程序中原来的指令顺序预取、译码和分发，但不按原来的指令顺序完成指令的执行。

无序发射无序完成

不按照程序中原来的指令顺序预取、译码和分发，也不按原来的指令顺序完成指令的执行。

写后读(RAW)相关

若前一条指令写入某寄存器，后一条指令需要从该寄存器读出，则称这两条指令是写后读相关的。

写后写(WAW)相关

若前一条指令和后一条指令都需要对同一个寄存器进行写入，则称这两条指令是写后写相关的。这种相关性，在无序发射或乱序执行时可能会发生数据冒险，而在按序发射并按序完成时，不会发生数据冒险。

7.4 常见问题解答

1. 什么样的指令格式更适合流水线方式?

答：定长指令字和定长操作码使得每条指令的预取和译码操作一致，便于流水线控制；指令类型少、操作数地址规整便于规划取操作数步骤，并使得对指令进行译码的同时，可以读取寄存器操作数；采用 Load/Store 型指令风格便于利用执行运算步骤来进行地址计算。由此可知，RISC 指令设计风格更适合流水线方式。

2. 采用流水线方式能使一条指令的执行时间更短吗?

答：不能。采用流水线方式使得指令吞吐率提高了，即在给定的时间内完成指令执行的条数增加了，但每条指令的执行过程没有减少，所以不会缩短一条指令的执行时间，反而会延长一条指令的执行时间。

3. 为什么流水线方式会延长一条指令的执行时间?

答：因为在确定一条流水线的流水段个数时，是以最复杂指令执行过程所需要的流水段个数为标准设计的；在确定每个流水段的宽度时，也以最复杂流水段所需要的宽度来设计。因而，所有指令都需要花费最慢指令所需要的执行时间才能完成执行。此外，每个流水段之间要有信息的缓存和传递等，这也增加了额外的执行时间开销。

4. 二阶段流水线能成倍提高指令执行效率吗?

答：不能。二阶段流水线把一个指令周期分成取指令和执行指令两个阶段，是最简单

的流水线，它不能成倍提高指令执行效率。这有很多原因，主要有：(1)因为每条指令的执行阶段所花时间不同，流水线要求以最慢指令为标准来设计，所以，执行阶段时间可能会很长。(2)虽然每条指令的取指令过程可能一样，但是流水线中要求每个流水段的时间相同，所以，取指令阶段时间也要和最慢指令的执行时间一样。(3)流水段寄存器会增加一定的延迟。(4)各种流水线冒险会破坏流水线，造成流水线的停顿，影响指令执行效率。

5. 加倍流水线的阶段数能成倍提高指令执行效率吗？

答：不能。一般来说，加倍流水线的阶段数会提高指令的执行效率，但是不能成倍提高。主要原因是因为增加流水段，使得流水段之间的流水段寄存器的读写开销增加了。例如，假定一个三阶段流水线的每一级流水段中组合逻辑延时为100ps，流水段寄存器延时为20ps，则时钟周期至少为100＋20＝120ps，吞吐率为1/120ps＝8.33GOPS(每秒千兆次操作)，指令延时至少为3×120＝360ps；如果流水段数成倍增加到六段，则每个流水段的组合逻辑延时变为50ps，故时钟周期变为50＋20＝70ps，吞吐率为1/70ps＝14.29GOPS，指令延时为6×70＝420ps。所以，性能只提高到大约14.29/8.33＝1.71倍，而不是2倍，指令延时增加了420－360＝60ps，这主要是流水段寄存器增加的延时。

6. 流水线深度越深，时钟频率就越高，对吗？

答：一般来说，在指令执行时间一定的情况下，流水线深度越深，时钟频率就越高。增加流水段个数，使得每个流水段内的操作变得非常简单，因而，每个阶段的延时就很小，也就缩短了时钟周期，提高了时钟频率。但是，流水线深度不能无限制提高，随着流水段个数的增加，流水段之间的流水段寄存器增多，加大了流水段之间缓冲的额外开销，当额外开销的比例达到50%时，再增加流水线的深度就没有意义了。此外，用于流水线优化和存储器(或寄存器)冲突处理的控制逻辑将随流水线深度的加深而大量增多，可能导致用于流水段之间的控制逻辑比流水段本身的控制逻辑更复杂。

7. 流水线方式下，如何确定流水段的个数？

答：流水线方式下，一条指令的执行过程被分成了若干个操作子过程。由于每条指令所完成的功能不同，所包含的操作过程就不同。有的指令完成寄存器之间的传送；有的是简单的加/减运算；还有的是复杂的乘/除运算。这些操作所花的时间相差很大，因此，这些指令如果都在同一个流水线中执行，就必须按最复杂的指令来设计流水线的流水段个数。现代计算机一般把复杂度相近的指令用同一条流水线完成，而把复杂度相差很大的指令安排在不同的流水线中。

8. 流水线方式执行指令时，总能在一个时钟内完成一条指令的执行吗？

答：不能。理想情况下，经过若干周期后，能在每个周期内完成一条指令，即CPI＝1。但是，当程序中出现以下情况时，流水线被破坏，因而，不能达到CPI＝1。(1)当有多条指令的不同阶段都要用到同一个功能部件时，发生资源冲突，后面指令要延时执行；(2)当程序的执行流程发生改变时，发生控制相关，原来按顺序取出的指令无效；(3)当后面指令的操作数是前面指令的运行结果时，发生数据相关，后面指令要延时执行。此外，还有cache缺失、TLB缺失、异常和中断等的发生都会阻塞流水线的执行。

9. 如何解决结构冒险？

答：通过以下两种规整流水线结构的方式可以解决部分结构冒险。(1)规定每个功能部件在每条指令执行过程中只能被使用一次，例如，每条指令只能用一次“寄存器写口”；

(2)每个功能部件只能在一个特定的流水段内被使用,例如,“寄存器写口”只能在第5个流水阶段被使用。

另外,指令和数据分别存放在不同的存储器中,使得同时访问指令和数据不会引起存储器资源的结构冒险。这在大多数处理器中都是这样的,因为在L1 cache的data cache和code cache是分开的。

10. 什么是控制冒险?哪些情况下会发生控制冒险?

答: 正常情况下PC的值按顺序增量,但在执行转移类指令或发生异常和中断时,PC的值由指令或异常/中断处理部件给出。流水线方式下,如果在取下一条指令时,下一条指令的地址还没有送到PC中,那么所取的下一条指令就不是正确的,因而发生控制冒险。可以看出,如果确定下一条指令地址所用时间较长,就会因为来不及在一个时钟周期内得到正确的PC值而发生控制冒险。通常,转移类指令会发生控制冒险,例如,分支指令(条件转移指令)要根据条件测试结果来确定PC的值,因而会发生控制冒险;返回指令需要从存储器中读取返回地址送PC,因而也会发生控制冒险。

11. 无条件转移(如跳转指令、调用指令等)是否可以避免发生控制冒险?

答: 可能会发生控制冒险。如果不是把取指令操作和译码操作合成一个阶段,就会引起控制冒险。因为要等到译码阶段才能知道是转移指令,而此时流水线中正在取下一条顺序执行的指令,即使译码阶段能够得到转移目标地址,也已经有一条不该执行的指令在流水线中,即流水线被阻塞了一个时钟周期。在下一个时钟到达时,应把已经取出的下一条指令清“0”,并把转移目标地址送PC,重新从转移目标地址处取指令执行。

12. 流水线中如何控制在同一时钟内各流水段中执行不同的指令?

答: 对于每条指令来说,流水线中前面的取指令流水段和指令译码流水段,都是一样的。所以这两个流水段中的操作没有必要和哪个指令对应。但是,指令译码以后,每个流水段中执行的动作一定要和特定的指令对应,也就是说,特定指令对应的控制信号一定要送到对应的流水段中,以控制该流水段中的执行部件完成正确的动作。这样也就可以控制在同一时钟内各流水段执行不同的指令。要做到特定指令对应的控制信号送到对应的流水段中,只要将译码阶段得到的控制信号也以流水线的方式传送,每来一个时钟,控制信号就往后面一个流水段传送一次,这样使得控制信号和所控制的操作同步。

13. 指令译码前控制信号还没有产生,那么如何控制译码前指令的动作呢?

答: 任何指令总是先要取指令,然后对指令译码,最后根据指令的不同产生不同的控制信号,控制数据通路完成不同的指令功能。也就是说,在指令译码前控制信号还没有产生,所以,不可能有控制信号来控制译码前指令的动作,而且也不需要控制信号来控制译码前指令的动作,其原因是因为这些动作每条指令都一样,是确定的。取指令/PC+4、译码/读寄存器,而这些操作都可以不在控制信号的控制下进行。

7.5 单项选择题

1. 以下关于指令流水线设计的叙述中,错误的是(　　)。

   A. 指令执行过程中的各个子功能都须包含在某个流水段中

   B. 所有子功能都必须按一定的顺序经过流水段

C. 虽然各子功能所用实际时间可能不同，但经过每个流水段的时间都一样

D. 任何时候各个流水段的功能部件都不可能执行空(nop)操作

2. 以下关于流水段寄存器的叙述中，正确的是(　　)。

A. 指令译码得到的控制信号需通过流水段寄存器传递到后面的流水段

B. 每个流水段之间的流水段寄存器位数一定相同

C. 每个流水段之间的流水段寄存器存放的信息一定相同

D. 用户程序可以通过指令指定访问哪个流水段寄存器

3. 以下关于指令流水线和指令执行效率关系的叙述中，错误的是(　　)。

A. 加倍增加流水段个数不能成倍提高指令执行效率

B. 随着流水段个数的增加，流水段之间缓存开销的比例增大

C. 加深流水线深度，可以提高处理器的时钟频率

D. 为了提高指令吞吐率，流水段个数可以无限制地增多

4. 以下有关流水线数据通路的描述中，错误的是(　　)。

A. 每个流水段由执行指令子功能的功能部件和流水段寄存器组成

B. 控制信号仅作用在功能部件上，时钟信号仅作用在流水段寄存器上

C. 在没有阻塞的情况下，PC的值在每个时钟周期都会改变

D. 取指令阶段和指令译码阶段不需要控制信号的控制

5. 某计算机的指令流水线由4个功能段组成，指令流经各功能段的时间(忽略各功能段之间流水段寄存器的缓存时间)分别为90ns、80ns、70ns和60ns，则该计算机的CPU时钟周期至少是(　　)。

A. 90ns　　B. 80ns　　C. 70ns　　D. 60ns

6. 假定执行最复杂的指令需要完成6个子功能，分别由对应的功能部件A～F来完成，每个功能部件所花的时间分别为80ps、40ps、50ps、70ps、20ps、30ps，流水线寄存器延时为20ps，现把最后两个功能部件E和F合并，以产生一个五段流水线。该五段流水线的时钟周期至少是(　　)ps。

A. 70　　B. 80　　C. 90　　D. 100

7. 以下有关流水段的功能部件的描述中，错误的是(　　)。

A. 所有功能部件都是用组合逻辑实现

B. 同一个功能部件可以在不同的流水段中被使用

C. 每个功能部件在每条指令中都只被用一次

D. 寄存器写口只能在指令结束时的“写回”阶段被用

8. 以下给定的情况中，不会引起指令流水线阻塞的是(　　)。

A. 访存冲突　　B. 指令数据相关

C. 执行空操作指令　　D. cache缺失

9. 以下给定的情况中，不会引起指令流水线阻塞的是(　　)。

A. 数据旁路(转发)　　B. TLB缺失

C. 条件转移　　D. 外部中断

10. 以下是关于结构冒险的叙述：

① 结构冒险是指同时有多条指令使用同一个资源

② 避免结构冒险的基本做法是使每个指令在相同流水段中使用相同的部件
③ 重复设置功能部件可以避免结构冒险
④ 数据 cache 和指令 cache 分离可解决同时访问数据和指令的冒险
以上叙述中，正确的有(　　)。
A. ①、②、④　　B. ①、②、③　　C. ①、③、④　　D. 全部

11. 以下是关于数据冒险的叙述：
① 数据冒险是指后面指令用到的数据还未来得及由前面指令产生
② 在发生数据冒险的指令之间插入空操作指令能避免数据冒险
③ 采用转发(旁路)技术可解决部分数据冒险
④ 通过编译器调整指令顺序可解决部分数据冒险
以上叙述中，正确的有(　　)。
A. ①、②、④　　B. ①、②、③　　C. ①、③、④　　D. 全部

12. 以下是关于控制冒险的叙述：
① 条件转移指令执行时可能会发生控制冒险
② 在分支指令后加入若干空操作指令可避免控制冒险
③ 采用转发(旁路)技术可以解决部分控制冒险
④ 通过编译器调整指令顺序可解决部分控制冒险
以上叙述中，正确的有(　　)。
A. ①、②、④　　B. ①、②、③　　C. ①、③、④　　D. 全部

13. 以下是有关分支预测的叙述：
① 分支预测技术可用于控制冒险和数据冒险处理
② 采用静态预测技术时，每次的预测结果总是一样
③ 通常情况下，动态预测比静态预测的预测成功率高
④ 预测错误时已被错误地取到流水线执行的指令必须被冲刷掉
以上叙述中，正确的有(　　)。
A. ①、②、③　　B. ①、②、④　　C. ②、③、④　　D. 全部

14. 以下是一段 MIPS 指令序列：

```
1   loop:   add  $t1, $s3, $s3      #R[$t1]←R[$s3]+R[$s3]
2           add  $t1, $t1, $t1      #R[$t1]←R[$t1]+R[$t1]
3           lw   $t0, 0($t1)        #R[$t0]←M[R[$t1]+0]
4           bne  $t0, $s5, exit     #if (R[$t0]≠R[$s5]) then go to exit
5           add  $s3, $s3, $s4      #R[$s3]←R[$s3]+R[$s4]
6           j    loop               #go to loop
7   exit:
```

以上指令序列中，第(　　)行指令产生了一个分支控制冒险。
A. 2　　B. 4　　C. 5　　D. 7

15. 以下是一段 MIPS 指令序列：

```
1           add  $t1, $t0, $t1      #R[$t1]←R[$t0]+R[$t1]
2           lw   $t0, 0($t1)        #R[$t0]←M[R[$t1]+0]
```

```
3         bne   $t0, $s5, exit          #if(R[$t0]≠R[$s5]) then go to exit
4         add   $s3, $s5, $s4           #R[$s3]←R[$s5]+R[$s4]
5   exit:
```

以上指令序列中,(　　)指令之间产生数据相关。

A. 1和2、2和3　　B. 1和2、2和3、3和4

C. 1和2、1和3　　D. 1和2、1和3、2和3

16. 以下是一段MIPS指令序列:

```
1         addi  $t1, $zero, 20          #R[$t1]←20
2         lw    $t2, 12($a0)            #R[$t2]←M[R[$a0]+12]
3         add   $v0, $t1, $t2           #R[$v0]←R[$t1]+R[$t2]
```

以上指令序列中,第1和第3、第2和第3条指令之间发生数据相关。假定采用"取指、译码/取数、执行、访存、写回"这种五段流水线方式,并控制在时钟周期的前半周期写寄存器堆后半周期读寄存器堆,那么不采用"转发"技术时,需要在第3条指令前加入(　　)条空操作(nop)指令才能使这段程序不发生数据冒险。

A. 1　　B. 2　　C. 3　　D. 4

17. 以下是一段MIPS指令序列:

```
1         addi  $t1, $zero, 20          #R[$t1]←20
2         lw    $t2, 12($a0)            #R[$t2]←M[R[$a0]+12]
3         add   $v0, $t1, $t2           #R[$v0]←R[$t1]+R[$t2]
```

以上指令序列中,第1和第3、第2和第3条指令之间发生数据相关。假定采用"取指、译码/取数、执行、访存、写回"这种五段流水线方式,那么在采用"转发"技术时,需要在第3条指令前加入(　　)条空操作(nop)指令才能使这段程序不发生数据冒险。

A. 0　　B. 1　　C. 2　　D. 3

18. 以下有关超标量技术的叙述中,错误的是(　　)。

A. 超标量技术是指在流水线中采用更多的流水段个数

B. 超标量方式执行指令时可以同时发射多条指令至流水线中

C. 采用超标量技术的CPU中必须配置多个不同的功能部件

D. 采用超标量技术的目的是利用部件的并行性以提高指令吞吐率

【参考答案】

1. D　2. A　3. D　4. B　5. A　6. D　7. B
8. C　9. A　10. D　11. D　12. A　13. C　14. B
15. A　16. B　17. B　18. A

7.6 分析应用题

1. 假定某计算机工程师想设计一个新CPU,其中运行的一个典型程序的核心模块有一百万条指令,每条指令执行时间为100ps。

(1) 在非流水线处理器上执行该程序大约需要花多长时间？

(2) 若新 CPU 是一个 20 级流水线处理器，执行上述同样的程序，理想情况下，它比非流水线处理器快多少？

(3) 实际流水线并不是理想的，流水段之间的数据传送会有额外开销。这些开销是否会影响指令执行时间(Instruction Latency)和指令吞吐率(Instruction Throughput)？

【分析解答】

(1) 非流水线处理器上执行该程序的时间大约为 $100\text{ps}\times10^6=100\mu\text{s}$。

(2) 若在一个 20 级流水线的处理器上执行，忽略流水段之间的寄存器延时，理想情况下，每个时钟周期为 100/20＝5ps，所以，程序执行时间大约为 $5\text{ps}\times10^6=5\mu\text{s}$，因此，大约快 100/5＝20 倍。

(3) 流水段之间数据的传递产生的额外开销，使得一条指令的执行时间被延长，即影响了指令执行时间；同时也延长了每个流水段的时间，即影响了指令吞吐率。

2. 某计算机指令流水线由 6 个功能段组成，依次为 A～F，每个功能段的组合逻辑延迟分别为 80ps、30ps、60ps、50ps、60ps、20ps，最后一个功能段需要写寄存器，寄存器延时为 20ps。在这些组合逻辑块之间插入必要的流水段寄存器就可实现相应的指令流水线。理想情况下，以下各种方式所得到的时钟周期、指令吞吐率和指令执行时间各是多少？应该在哪里插入流水段寄存器(假定插入的流水段寄存器的延时也为 20ps)？根据对以下 4 种情况的分析，你能得到什么结论？

(1) 插入一个流水段寄存器，得到一个两级流水线。

(2) 插入两个流水段寄存器，得到一个三级流水线。

(3) 插入三个流水段寄存器，得到一个四级流水线。

(4) 吞吐量最大的流水线。

【分析解答】

(1) 两级流水线的平衡点在 C 和 D 之间，其前面一个流水段的组合逻辑延时为 80＋30＋60＝170ps，后面一个流水段的组合逻辑延时为 50＋60＋20＝130ps。因而最长功能段延时为 170ps，加上流水段寄存器延时 20ps，因而时钟周期为 190ps，理想情况下，指令吞吐率为每秒钟执行 $1/(190\times10^{-12})=5.26$G 条指令。每条指令在流水线中的执行时间为 $2\times190=380$ps。

(2) 两个流水段寄存器分别插在 B 和 C、D 和 E 之间，这样第一个流水段的组合逻辑延时为 80＋30＝110ps，中间第二段的延时为 60＋50＝110ps，最后一个段延时为 60＋20＝80ps。这样，每个流水段所用时间都按最长延时调整为 110＋20＝130ps，故时钟周期为 130ps，指令吞吐率为每秒钟执行 $1/(130\times10^{-12})=7.69$G 条指令，每条指令在流水线中的执行时间为 $3\times130=390$ps。

(3) 3 个流水段寄存器分别插在 A 和 B、C 和 D、D 和 E 之间，这样第一个流水段的组合逻辑延时为 80ps，第二段延时为 30＋60＝90ps，第三段延时为 50ps，最后一段延时为 60＋20＝80ps。这样，每个流水段都以最长延时调整为 90＋20＝110ps，故时钟周期为 110ps，指令吞吐率为每秒钟执行 $1/(110\times10^{-12})=9.09$G 条指令，每条指令在流水线中的执行时间为 $4\times110=440$ps。

(4) 因为各功能部件对应的组合逻辑中最长延时为 80ps，所以，流水线的时钟周期肯定比 80ps＋20ps＝100ps 长。为达到最大吞吐率，时钟周期应尽量短，因此，最合理的划分方

案应按照每个时钟周期为100ps来进行。根据每个功能部件所用时间可知，流水线至少按5段来划分，分别把流水段寄存器插入A和B、B和C、C和D、D和E之间，这样各段的组合逻辑延时为80ps、30ps、60ps、50ps和80ps。其中，最后一个延时80ps由E和F两个阶段的时间相加得到。这样时钟周期为100ps，指令吞吐率为每秒钟执行$1/(100\times10^{-12})=10G$条指令，每个指令的执行时间为$5\times100=500ps$。

通过对上述4种情况进行分析，可以得出以下结论：划分的流水段多，时钟周期就变短，指令执行吞吐率就变高，而相应的额外开销（插入的流水段寄存器的延时）也变大，使得一条指令的执行时间变长。

3. 假定在如图7.1所示的五级流水线处理器中，各主要功能部件的操作时间如下。存储器：200ps；ALU或加法器：150ps；寄存器堆读口或写口：50ps。请回答下列问题：

(1) 若执行阶段EX所用的ALU操作时间缩短20%，则能否加快流水线执行速度？如果能，能加快多少？如果不能，为什么？

(2) 若ALU操作时间增加20%，对流水线的性能有何影响？

(3) 若ALU操作时间增加40%，对流水线的性能又有何影响？

【分析解答】

(1) ALU操作时间缩短20%不能加快流水线执行速度。因为指令流水线的执行速度取决于最慢的功能部件所用时间，最慢的是存储器，只有缩短了存储器的操作时间才可能加快流水线速度。

(2) ALU操作时间延长20%时，变成了180ps，比存储器所用时间200ps还小，因此，对流水线性能没有影响。

(3) ALU操作时间延长40%时，变成了210ps，比存储器所用时间200ps大，因此，在不考虑流水段寄存器延时的情况下，流水线的时钟周期从200ps变为210ps，流水线执行速度降低了$(210-200)/200=5\%$。

4. 下面是一段MIPS指令序列：

```
1       add     $t1, $s1, $s0
2       sub     $t2, $s0, $s3
3       add     $t1, $t1, $t2
```

假定在一个采用“取指、译码/取数、执行、访存、写回”的五段流水线处理器中执行上述指令序列，请回答下列问题：

(1) 以上指令序列中，哪些指令之间发生数据相关？

(2) 不采用“转发”技术的话，需要在何处、加入几条nop指令才能使这段指令序列的执行避免数据冒险？

(3) 如果采用“转发”技术，是否可以完全解决数据冒险？不行的话，需要在何处、加入几条nop指令才能使这段指令序列的执行避免数据冒险？

【分析解答】

(1) 因为第1条和第2条指令都会更新第3条指令用到的寄存器的值，有可能导致第3条指令取操作数时得到的是更新前的数据，这样第3条指令就不能正确执行，因此，第1条和第3条指令、第2条和第3条指令之间发生的数据相关。

（2）不进行“转发”的话，就只能通过在第 3 条指令前加 nop 指令来延迟第 3 条指令的执行。因为只有第 2 条指令把数据写回到 $ t2，第 3 条指令才能从 $ t2 取到正确的值，所以，第 2 条指令的“写回（Wr）”流水段后面才应该是第 3 条指令的“译码/取数（ID）”流水段，为此，在第 2 条和第 3 条指令之间必须插入 3 条 nop 指令，如图 7.5 所示。

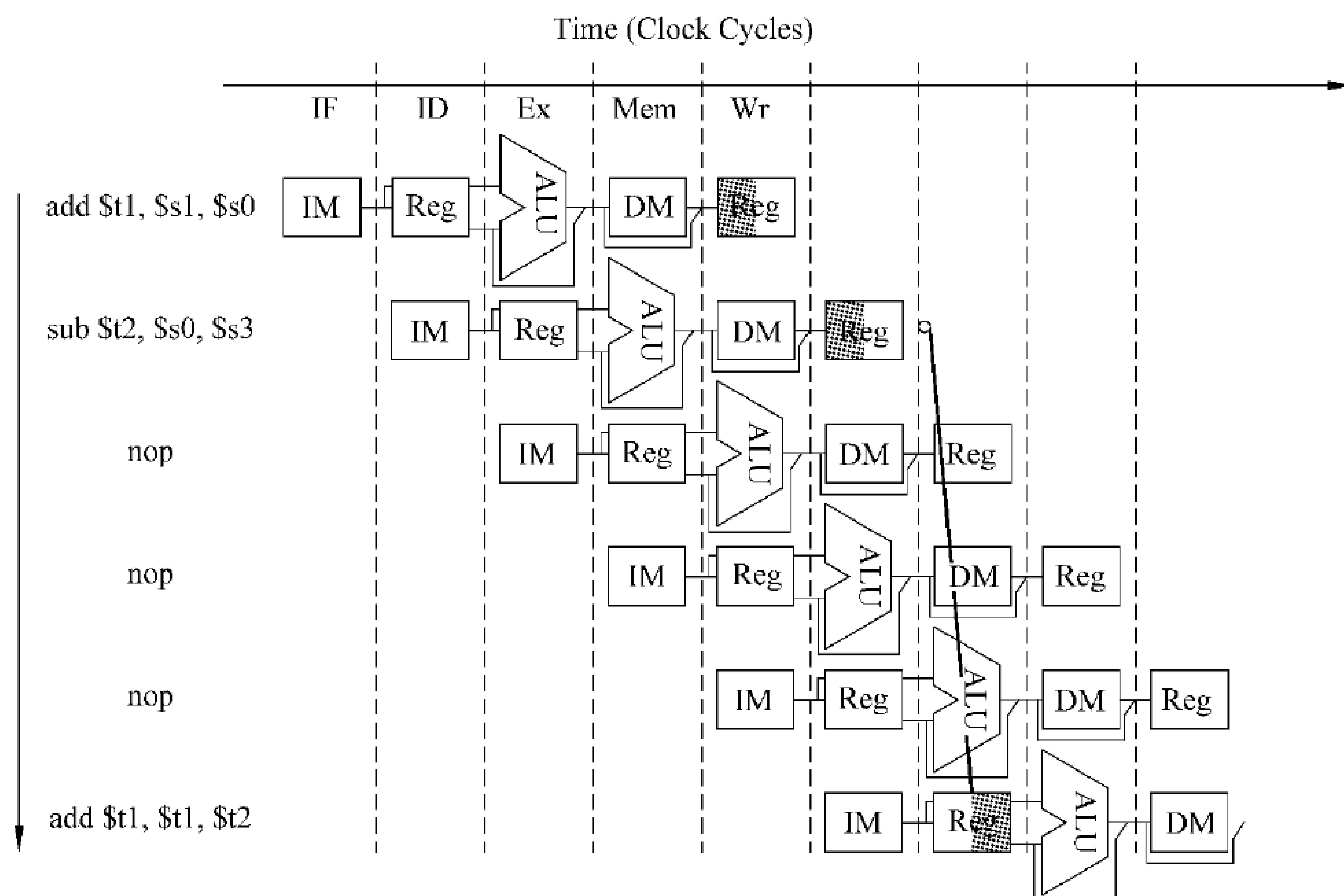


图 7.5　题 4 的图示 1

若将寄存器写口和寄存器读口分别安排在一个时钟周期的前、后半个周期内独立工作，使得前半周期写入寄存器的内容在后半周期能够正确读出，那么，只要加入两条 nop 指令即可，如图 7.6 所示。

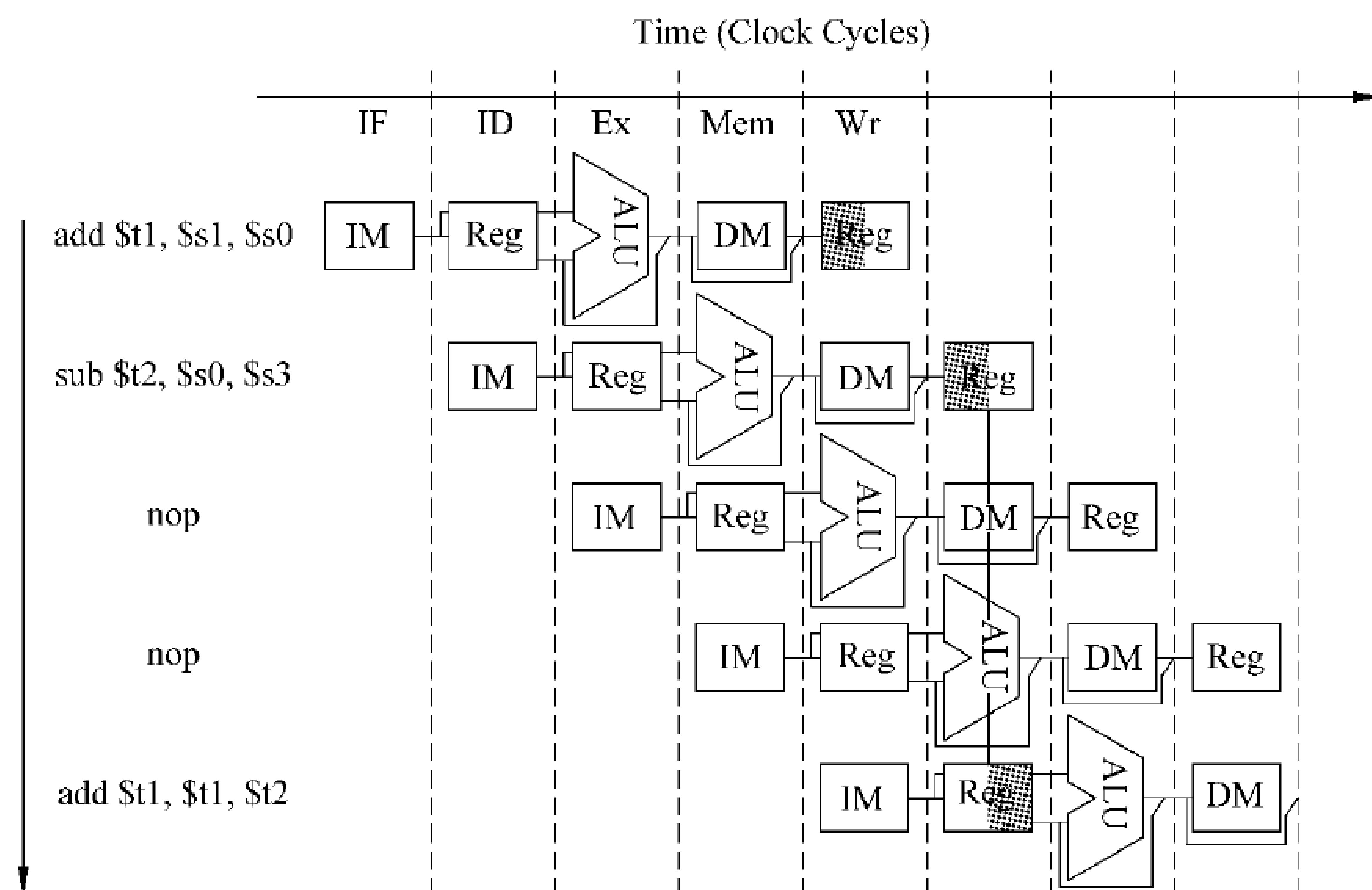


图 7.6　题 4 的图示 2

(3) 采用"转发"技术，上述程序段可以完全避免数据冒险。只要把第1条指令"访存(Mem)"段结束时在流水段寄存器中的$t1的值和第2条指令"执行(Ex)"段结束时在流水段寄存器中的$t2的值同时"转发"到第3条指令的"执行(Ex)"段内ALU的两个输入端，这样，在ALU中运算的两个操作数都是正确的值，不会发生数据冒险，无须再插入nop指令，如图7.7所示。

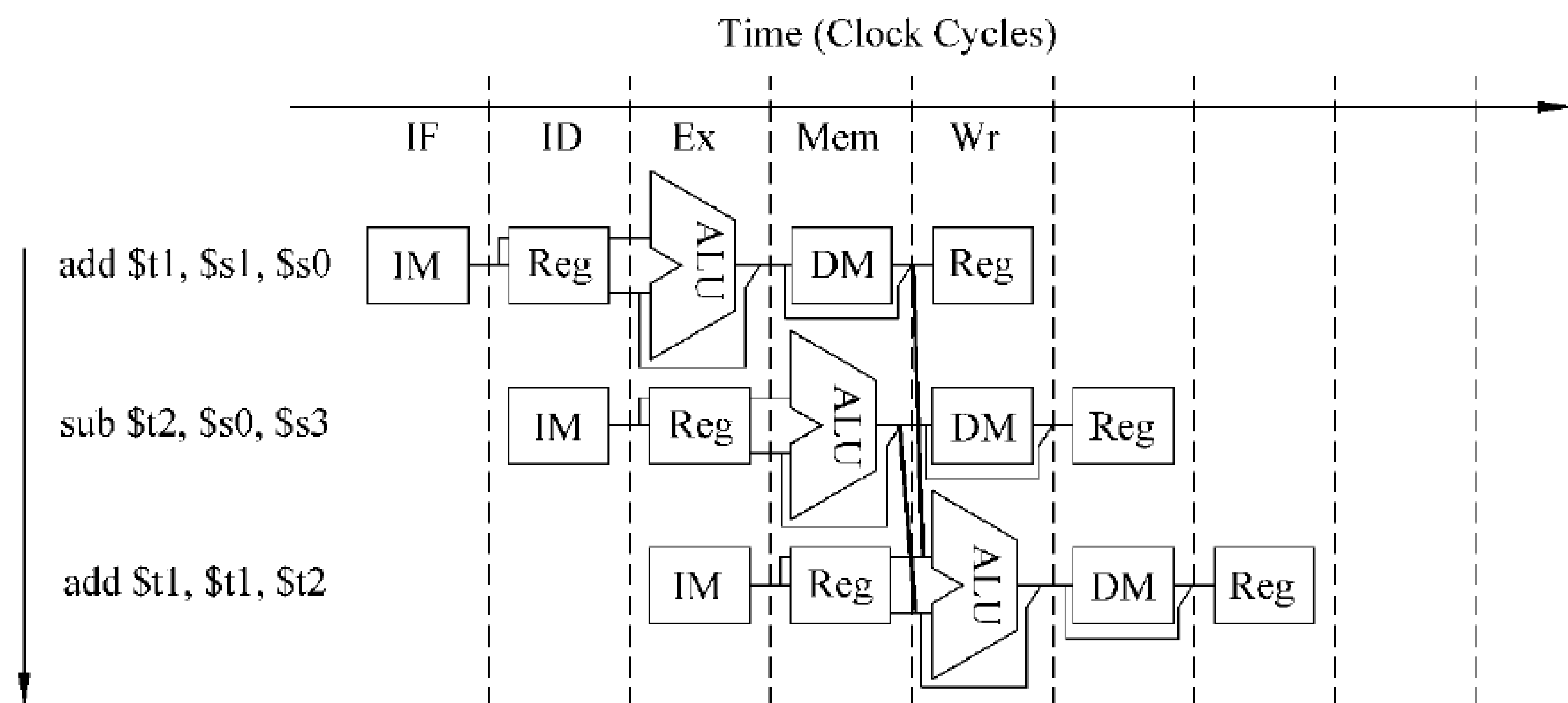


图7.7　题4的图示3

5. 下面是一段MIPS指令序列：

```
1        add    $s3, $s1, $s0
2        add    $t2, $s0, $s3
3        lw     $t1,0($t2)
4        add    $t1, $t1, $t2
```

假定在一个采用"取指、译码/取数、执行、访存、写回"的五段流水线处理器中执行上述指令序列，该流水线数据通路中，寄存器写口和寄存器读口分别安排在一个时钟周期的前、后半个周期内独立工作。请回答下列问题：

(1) 以上指令序列中，哪些指令之间发生数据相关？

(2) 不采用"转发"技术的话，需要在何处、加入几条nop指令才能使这段指令序列的执行避免数据冒险？nop指令增加的百分比为多少？该指令序列的执行共需要多少时钟周期？

(3) 如果采用"转发"技术，是否可以完全解决数据冒险？不行的话，需要在何处、加入几条nop指令才能使这段指令序列的执行避免数据冒险？

(4) 若数据冒险通过硬件阻塞进行处理，则在不采用"转发"和采用"转发"两种情况下，执行上述4条指令的CPI分别是多少？

【分析解答】

(1) 发生数据相关的是：第1条和第2条指令之间关于$s3，第2条和第3条指令之间关于$t2，第2条和第4条指令之间关于$t2，以及第3条和第4条指令之间关于$t1。

(2) 不进行"转发"处理的话，需要分别在第2、3、4条指令前加2条nop指令才能避免数据冒险，共加了6条nop指令。nop指令增加的百分比为6/4＝150%，即平均1条指令加1.5条nop指令。该指令序列执行所需要的时钟周期数为(5－1)＋(4＋6)＝14。

(3) 通过"转发"可以避免第1和第2、第2和第3、第2和第4条指令之间的数据相关；

但第 3 和第 4 条指令之间是 Load-use 数据相关，因此，无法用“转发”消除冒险，而需在第 4 条指令前加入 1 条 nop 指令。

(4) 数据冒险通过硬件阻塞进行处理时，如不采用“转发”来执行上述 4 条指令，则需要的时钟周期数为 14，故 CPI 为 14/4=3.5。若采用“转发”来执行上述 4 条指令，则需要时钟周期数为(5−1)+(4+1)=9，故 CPI 为 9/4=2.25。

6. 下面是一段 MIPS 指令序列：

```
1        add     $t1, $s1, $s0
2        sub     $t2, $s0, $s3
3        lw      $t0, 0($t2)
4        add     $s0, $t0, $s2
5        add     $t1, $s0, $t2
```

假定在一个采用“取指、译码/取数、执行、访存、写回”的五段流水线处理器中执行上述指令序列，并且该处理器采用了“转发”和“Load-use 冒险处理”技术，则在第 5 个时钟周期内，各指令的执行情况如何？哪些寄存器的内容正在被读？哪些寄存器将被写入数据？

【分析解答】

在一个如图 7.4 所示的带“Load-use 冒险处理”和“转发”技术的五阶段流水线数据通路中执行上述指令序列，则第 5 个时钟周期内各条指令的执行情况如下。

第 1 条指令在“写回(Wr)”阶段，寄存器 $ t1 将被写入。

第 2 条指令在“访存(Mem)”阶段，sub 指令在该阶段进行的是空操作；在转发检测单元中，因为流水段寄存器 Ex/Mem 中的目的寄存器 RegRd 为 $ t2，流水段寄存器 ID/Ex 中的源寄存器 Rs 也为 $ t2，同时，流水段寄存器 Ex/Mem 中的 RegWr 控制信号为 1，所以检测到转发条件满足，因而，此时，sub 指令在上一个时钟周期中的执行结果(在流水段寄存器 Ex/Mem 中的 ALU 输出结果)正被回送到 ALU 的输入端。

第 3 条指令在“执行(Ex)”阶段，ALU 正在执行 add 操作，进行地址运算，ALU 输出结果将被写入流水线寄存器 Ex/Mem 中。

第 4 条指令在“译码/取数(ID)”阶段，寄存器 $ t0 和 $ s2 的内容正被读出；在 Load-use 冒险检测单元中，因为流水段寄存器 IF/ID 中源操作数寄存器 Rs 为 $ t0，流水段寄存器 ID/Ex 中目的操作数寄存器 Rt 也为 $ t0，同时，因为上一条指令是 lw，故流水段寄存器 ID/Ex 中的 MemRead 控制信号为 1，所以在该阶段检测到 Load-use 冒险条件满足，此时，需要进行 Load-use 冒险处理，在流水线中插入一个“气泡”，将指令的执行阻塞一个时钟周期。包括以下 3 个步骤：(1)将流水段寄存器 ID/Ex 中的控制信号全部清“0”，以保证第 4 条指令被阻塞一个时钟周期执行；(2)将流水段寄存器 IF/ID 中的指令维持不变，以保证第 4 条指令重新译码后执行；(3)将 PC 的值维持不变，以保证根据 PC 的值重新取出第 5 条指令。

第 5 条指令在“IF”阶段，指令正被读出，将要送到流水段寄存器 IF/ID 的输入端。因为之前发生了 Load-use 数据冒险，所以该指令将在随后的第 6 个时钟周期内重新被读出。

7. 以下是一段 MIPS 指令序列：

```
1    loop:    add     $t1, $s3, $s3
2             add     $t1, $t1, $t1
3             add     $t1, $t1, $s6
```

```
4            lw      $t0, 0($t1)
5            bne     $t0, $s5, exit
6            add     $s3, $s3, $s4
7            j       loop
8    exit:
```

假定在一个采用"取指、译码/取数、执行、访存、写回"的五段流水线中执行上述指令序列，该流水线数据通路中，寄存器写口和寄存器读口分别安排在一个时钟周期的前、后半个周期内独立工作。要求回答下列问题：

(1) 哪些指令之间发生数据相关？哪些指令的执行会发生控制相关？

(2) 如果不采用"转发"技术进行数据冒险处理，那么应该在何处、加入几条 nop 指令才能避免数据冒险？假定采用"转发"技术处理，是否可以完全解决数据冒险？不行的话，需要在发生数据相关的指令前加入几条 nop 指令，才能使这段指令序列的执行避免数据冒险？

(3) 对于第 5 条分支指令引起的控制冒险(分支冒险)，假定将检测结果是否为"零"并更新 PC 的操作放在"访存(Mem)"阶段进行，则分支延迟损失时间片(分支延迟槽)为多少？在何处、加入几条 nop 指令可以消除分支冒险？若检测结果是否为"零"并更新 PC 的操作在"执行(Ex)"阶段进行，则分支延迟损失时间片(分支延迟槽)为多少？

(4) 对于第 7 条指令"j loop"，假定更新 PC 的操作在"执行(Ex)"阶段进行，则流水线会被阻塞几个时钟周期？需要在何处、加入几条 nop 指令才可消除该控制冒险？假定更新 PC 的操作在"译码(ID)"阶段进行，则流水线又将被阻塞几个时钟周期？

【分析解答】

(1) 发生数据相关的是：第 1 和第 2 条指令之间关于 $t1，第 2 和第 3 条指令之间关于 $t1，第 3 和第 4 条指令之间关于 $t1，第 4 和第 5 条指令之间关于 $t0，以及第 6 和第 1 条指令之间关于 $s3。此外，第 5 和第 7 条指令的执行都会发生控制相关。

(2) 对于数据冒险，如果不采用"转发"，而是简单地通过加入 nop 指令来避免冒险，那么应该在第 2、3、4、5 条指令前各加 2 条 nop 指令，以消除数据相关；对于第 6 条和第 1 条指令之间的数据相关，则可通过在第 7 条"j loop"指令后面加一条或两条 nop 指令消除(这样同时还能解决第 7 条"j loop"指令的控制冒险)；为解决数据冒险，共需加 13 或 14 条 nop 指令，大大增加了程序的长度，并极大地降低了指令执行效率。数据冒险在多数情况下可通过"转发"来解决，此处，第 2、3、4 条指令所需要的操作数可通过"转发"得到，无须加 nop 指令。第 5 条指令所需要的操作数 $t0 是 Load-use 冒险，不能用"转发"解决问题，需要在第 5 条指令前加 1 条 nop 指令，或通过硬件将第 5 条指令的执行阻塞 1 个时钟周期。

(3) 对于分支冒险，若检测结果是否为"零"并更新 PC 的操作在"访存(Mem)"阶段进行，则分支延迟损失时间片(分支延迟槽)为 3，此时在第 5 条分支指令后加 3 条 nop 指令，或从硬件上使分支指令后面一条指令的执行阻塞 3 个时钟周期。若检测结果是否为"零"并更新 PC 的操作在"执行(Ex)"阶段进行，则分支延迟损失时间片(分支延迟槽)为 2。

(4) 假定 j 指令更新 PC 的操作在"执行(Ex)"阶段进行，则流水线将被阻塞 2 个时钟周期，此时，需要在 Jump 指令后加 2 条 nop 指令才能消除该控制冒险。假定更新 PC 的操作在"译码(ID)"阶段进行，则流水线只被阻塞 1 个时钟周期。

8. 假定有一个程序共 1000 条指令，其指令序列为"lw, add, lw, add, …"。add 指令

仅依赖它前面的 lw 指令，而 lw 指令也仅依赖它前面的 add 指令。寄存器写口和寄存器读口分别在一个时钟周期的前、后半个周期内独立工作。请回答下列问题：

(1) 在带转发的 5 段流水线中执行该程序，其 CPI 为多少？

(2) 在不带转发的 5 段流水线中执行该程序，其 CPI 为多少？

【分析解答】

(1) 若流水线中有"转发"逻辑，并且寄存器写口和寄存器读口分别在一个时钟周期的前、后半个周期内工作，则只存在 lw 指令和 add 指令之间的一个 Load-use 数据冒险，因此，每个 lw 指令和 add 指令之间有一次流水线阻塞，即每一对 lw 和 add 指令需要用 3 个时钟周期完成，因而 CPI 为 1.5。

(2) 若流水线中没有"转发"逻辑，而寄存器写口和寄存器读口分别在一个时钟周期的前、后半个周期内工作，则在每两条相邻指令之间都会有两个阻塞，这样每条指令相当于都要有 3 个时钟才能完成，因而 CPI 为 3。

9. 假定在一个如图 7.4 所示的带"转发"功能的 5 段流水线中执行以下指令序列，则怎样调整以下指令序列才能使其性能达到最好？

```
1       lw   $2, 100($6)
2       add  $2, $2, $3
3       lw   $5, 200($7)
4       add  $6, $4, $7
5       sub  $3, $4, $6
6       lw   $2, 300($8)
7       beq  $2, $8, Loop
```

【分析解答】

因为采用"转发"技术，所以，只要对 Load-use 数据冒险进行指令序列调整。从上述指令序列来看，第 1 和第 2 条指令、第 6 和第 7 条指令之间存在 Load-use 数据冒险，所以，可将与第 2 和第 3 条指令无关的第 4 条指令插入第 2 条指令之前；将无关的第 5 条指令插入第 7 条指令之前。调整顺序后的指令序列如下。

```
1       lw   $2, 100($6)
4       add  $6, $4, $7
2       add  $2, $2, $3
3       lw   $5, 200($7)
6       lw   $2, 300($8)
5       sub  $3, $4, $6
7       beq  $2, $8, Loop
```

10. 假设将分支指令中的分支比较操作放到五段流水线的"译码/取数(ID)"阶段进行，那么，下列指令序列的执行过程中，哪些数据冒险不能通过"转发"技术解决？

```
1       lw   $1, 100 ($2)
2       addi $1, $1, 8
3       beq  $1, $3, 10
```

【分析解答】

如图 7.8 所示，给出的指令序列中，第 1 和第 2 条指令之间的 Load-use 数据冒险不能通过转发技术解决。对于第 2 和第 3 条指令之间的数据冒险，如果分支比较操作在“执行(Ex)”阶段以后进行，则完全可以将第 2 条指令的执行结果(在 Ex/Mem 流水段寄存器中)转发给到执行(Ex)阶段的 beq 指令的 ALU 输入端。但是，如果 beq 指令在 ID 阶段进行比较操作，因为在 ID 阶段的 beq 指令需要用到 \$1 的值时，第 2 条的 addi 指令还没有将 \$1 的值在 ALU 中计算出来，因此来不及转发，因而 beq 指令的执行需要阻塞一个时钟周期，如图 7.8 所示。

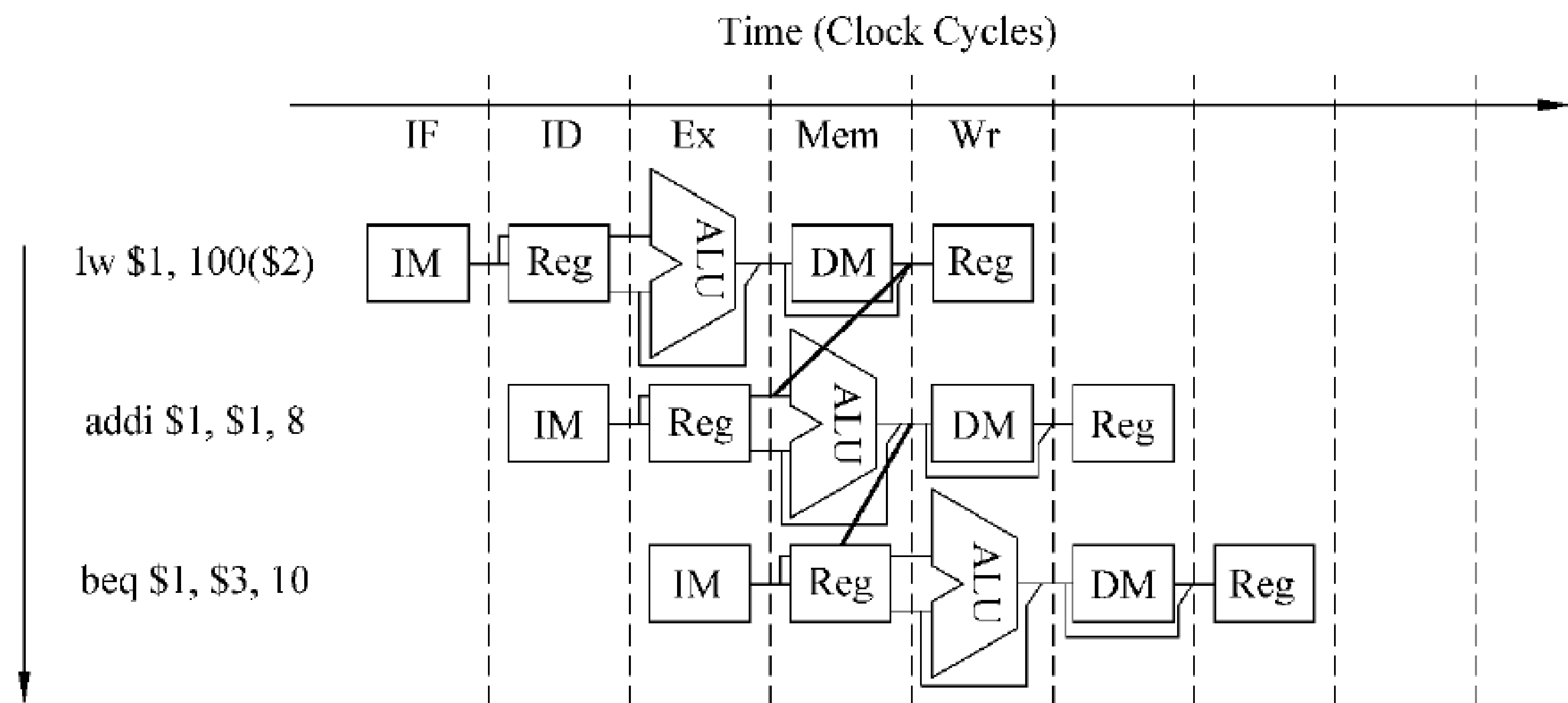


图 7.8　题 10 的图示

11. 假设数据通路中各主要功能部件的操作时间如下。存储器：200ps；ALU 和加法器：100ps；寄存器堆读口或写口：50ps。程序中指令的组成比例为取数 25%、存数 10%、ALU 52%、分支 11%、跳转 2%。假设非单周期处理器时的时钟周期取存储器存取时间的一半，MUX、控制单元、PC、扩展器和传输线路等的延迟都忽略不计，则下面的实现方式中，哪个最快？快多少？

(1) 单周期方式：每条指令在一个固定长度的时钟周期内完成。

(2) 多周期方式：每类指令的时钟数为取数-7，存数-6，ALU-5，分支-4，跳转-4。

(3) 流水线方式：采用取指 1、取指 2、取数/译码、执行、存取 1、存取 2、写回七段流水线；没有结构冒险；数据冒险采用“转发”技术处理；Load 指令与后续各指令之间存在依赖关系的概率分别为 1/2、1/4、1/8、…；分支延迟损失时间片为 2，分支预测准确率为 75%；不考虑异常、中断和访存缺失引起的流水线冒险。

【分析解答】

(1) 单周期方式下，时钟周期为 200+50+100+200+50=600ps，故一条指令的执行时间为 600ps。

(2) 多周期方式下，CPI=0.25×7+0.10×6+0.52×5+0.11×4+0.02×4=5.47，因为存储器操作变为在两个时钟周期内完成，所以多周期数据通路的时钟周期为 100ps，故平均一条指令的执行时间为 100×5.47=547ps。

(3) 流水线方式下，存储器操作变为在两个时钟周期内完成后，其流水线包含了 7 个阶段。对于分支指令，若预测正确，则不需额外时钟周期，故只需 1 个时钟周期；若预测错误，则因为分支延迟损失时间片为 2，所以应该将错误预取的 2 条指令冲刷掉，额外多用了 2 个

时钟周期，因此预测错误时共需 3 个时钟周期，故 CPI＝25％×3＋75％×1＝1.5。对于 Load 指令，因为一个存储操作占用两个时钟周期，所以随后第 1 条指令需 3 个(其中阻塞 2 个)时钟周期；随后第二条指令需 2 个(其中阻塞 1 个)时钟周期，以后的指令都不需要阻塞，故 CPI＝1/2×3＋1/4×2＋2/8×1＝2.25。对于 ALU 指令，随后的数据相关指令都可通过转发解决，故 CPI＝1；对于 Store 指令，不会发生数据冒险，故 CPI＝1；对于 Jump 指令，最快也要在译码阶段才能确定转移地址，故 CPI＝3。因此综合 CPI＝0.25×2.25＋0.10×1＋0.52×1＋0.11×1.5＋0.02×3＝1.41。所以，平均一条指令的执行时间为 1.41×100＝141ps。

由上述分析可知，流水线处理器的指令执行速度最快，是单周期的 600/141＝4.26 倍，是多周期的 547/141＝3.84 倍。

12. 假设有一段程序的核心模块中有 5 条分支指令，该模块将会被执行成千上万次，在其中一次执行过程中，5 条分支指令的实际执行情况如下(T：Taken；N：Not Taken)。

分支指令 1(B1)：T-T-T。

分支指令 2(B2)：N-N-N-N。

分支指令 3(B3)：T-N-T-N-T-N。

分支指令 4(B4)：T-T-T-N-T。

分支指令 5(B5)：T-T-N-T-T-N-T。

假定各个分支指令在每次模块执行过程中实际执行情况都一样，并且动态预测时，每个分支指令都有各自的预测表项，每次执行时的初始预测位都相同。请给出以下几种预测方案的预测准确率。

(1) 静态预测，总是预测转移 (Taken)。

(2) 静态预测，总是预测不转移(Not Taken)。

(3) 一位动态预测，初始预测转移 (Taken)。

(4) 二位动态预测，初始预测弱转移(Taken)。

【分析解答】

预测准确率＝预测正确次数/总预测次数×100％。以下 R 表示正确预测次数，W 表示错误预测次数。

(1) B1:R-3,W-0；B2:R-0,W-4；B3:R-3,W-3；B4:R-4,W-1；B5:R-5,W-2；60％。

(2) B1:R-0,W-3；B2:R-4,W-0；B3:R-3,W-3；B4:R-1,W-4；B5:R-2,W-5；40％。

(3) B1:R-3,W-0；B2:R-3,W-1；B3:R-1,W-5；B4:R-3,W-2；B5:R-3,W-4；52％。

(4) B1:R-3,W-0；B2:R-3,W-1；B3:R-3,W-3；B4:R-4,W-1；B5:R-5,W-2；72％。

13. 对于以下 MIPS 指令序列：

```
Loop:   lw   $2, 100($6)
        add  $3, $2, $7
        sw   $3, 100($6)
        lw   $5, 200($6)
        sub  $4, $5, $2
        sw   $4, 300($6)
        addi $6, $6, 8
        bne  $6, $8, Loop
```

假定在一个如图 7.4 所示的采用“转发”技术的五段流水线中执行上述循环 50 次，该流水线数据通路中，寄存器写口和寄存器读口分别被安排在一个时钟周期的前、后半个周期内独立工作，分支指令的分支延迟损失时间片为 3。要求回答下列问题：

(1) 不采用分支预测时，每次循环执行需要多少时钟周期？

(2) 若采用静态分支预测，并总是预测转移(Taken)，则各次循环执行需要多少时钟周期？总的执行时间比不采用分支预测时快多少？

(3) 如何调整指令顺序使得指令序列执行时流水线的阻塞次数最少？在不采用分支预测的情况下，指令顺序调整后总的执行时间少多少？在采用静态预测(总是预测转移)的情况下，指令顺序调整后总的执行时间少多少？

【分析解答】

(1) 循环中有 2 个 Load-use 冒险和一个控制(分支)冒险，因为分支指令的分支延迟损失时间片为 3，所以共有 2+3=5 次阻塞，因而每次循环的执行共需 8+5=13 个时钟周期。

(2) 若采用静态分支预测，并总是预测转移，则前面 49 次循环中的分支预测都能成功，每次循环需 8+2=10 个时钟周期，最后 1 次循环中的分支预测不成功，需要在流水线中冲刷掉 3 条指令，因而最后一次循环需 8+2+3=13 个时钟周期。这种情况下，总的执行时间为(5－1)＋49×10＋1×13＝507 个时钟周期。不采用分支预测时总的执行时间为(5－1)＋50×13＝654 个时钟周期。所以两者相比，采用静态预测比不采用分支预测时少用了 654－507＝147 个时钟周期。

(3) 可以将无关指令插入到 Load-use 数据相关指令之间，调整指令顺序后的指令序列如下。

```
Loop:  lw   $2, 100($6)
       lw   $5, 200($6)
       add  $3, $2, $7
       sw   $3, 100($6)
       sub  $4, $5, $2
       sw   $4, 300($6)
       addi $6, $6, 8
       bne  $6, $8, Loop
```

指令顺序调整后，消除了 Load-use 数据冒险，因而，在不采用分支预测的情况下，每次循环需 8+3=11 个时钟周期，共(5－1)＋11×50＝554 个时钟周期。比调整指令顺序前少了 654－554＝100 个时钟周期。

指令顺序调整后，在采用静态预测(总是预测转移)的情况下，前 49 次每次循环需 8 个时钟周期，最后 1 次预测错误，需在流水线中冲刷 3 条指令，因而需 8+3=11 个时钟周期，共需(5－1)＋49×8＋1×11＝407 个时钟周期，与调整指令顺序前相比，总的执行时间少了 507－407＝100 个时钟周期。

如果将指令序列优化并采用分支预测的情况与未进行指令序列优化且未采用分支预测的情况相比，则总的执行时间少了 654－407＝247 个时钟周期。

14. 某高级语言源程序中的一个 while 语句为“while (save[i]==k) i+=1;”，若对其编译时，编译器将 *i* 和 *k* 分别分配在寄存器 $s3 和 $s5 中，数组 *save* 的基址存放在 $s6

中,则生成的 MIPS 汇编代码段如下。

```
1  loop: sll  $t1, $s3, 2        #R[$t1]←R[$s3]<<2,即 R[$t1]=i×4
2        add  $t1, $t1, $s6      #R[$t1]←R[$t1]+R[$s6],即 R[$t1]=Address of save[i]
3        lw   $t0, 0($t1)        #R[$t0]←M[R[$t1]+0],即 R[$t0]=save[i]
4        bne  $t0, $s5, exit     #if R[$t0]≠ R[$s5] then goto exit
5        addi $s3, $s3, 1        #R[$s3]←R[$s3]+1,即 i=i+1
6        j    loop               #goto loop
7  exit:
```

假定如图 7.1 所示流水线数据通路中各主要功能单元的操作时间如下。存储器:200ps;ALU 和加法器:100ps;寄存器堆(读或写):50ps。请回答下列问题:

(1) 在不考虑流水段寄存器、多路选择器、控制单元、PC、扩展器和线路等延迟的情况下,五段流水线处理器的最小时钟周期为多少?

(2) 请指出循环体中指令之间的数据相关性。

(3) 假定采用“转发”技术,并对分支冒险采用“一位动态预测”(初始预测为转移)方式,条件检测和分支目标地址修改都在“执行(Ex)”阶段进行,Jump 指令在“译码(ID)”阶段进行跳转目标地址修改,则在流水线处理器上执行 8 次循环所用的时间为多少纳秒?对于同样的 8 次循环执行,与第 6 章的 6.6 节分析应用题 16 中的单周期和多周期处理器相比,流水线处理器的速度快了多少倍?

【分析解答】

(1) 因为最复杂的部件(存储器)所用的最长时间为 200ps,所以,在不考虑流水段寄存器、多路选择器、控制单元、PC、扩展器和线路等延迟的情况下,五段流水线处理器的最小时钟周期为 200ps。

(2) 循环体中第 1 和第 2、第 2 和第 3、第 3 和第 4、第 5 和第 1 条指令之间存在数据相关。

(3) 采用“转发”技术可以消除第 1 和第 2、第 2 和第 3、第 5 和第 1 条指令之间的数据相关,但不能消除第 3 和第 4 条指令之间的 Load-use 冒险,8 次循环共有 8 个时钟的阻塞;此外,对于 bne 控制(分支)冒险,第 1 次和最后 1 次预测错误,所以有 2 次需要对预取执行的指令进行冲刷。因为条件检测和转移目标地址修改都在“执行(Ex)”阶段进行,因此,分支延迟损失时间片(分支延迟槽)为 2,即每次冲刷掉 2 条指令。因此,2 次共被冲刷掉 4 条指令,使流水线阻塞了 4 个时钟周期;对于最后一条 Jump 指令,因为在“译码(ID)”阶段进行跳转目标地址修改,所以每次有一个时钟阻塞,8 次循环 Jump 指令共执行了 7 次,因而有7 个时钟周期的阻塞。综上可知,8 次循环总共有 $8+4+7=19$ 次阻塞,且第 1~4 条指令各执行了 8 次,第 5~6 条指令各执行了 7 次,因此,8 次循环所用的时钟周期数为 $(5-1)+4\times8+2\times7+19=69$,总共时间为 $69\times200\text{ps}=13.8\text{ns}$。

与第 6 章的 6.6 节分析应用题 16 的单周期处理器相比,该流水线处理器执行上述程序段的速度快了 $(27.6-13.8)/13.8=1$ 倍;与第 6 章的 6.6 节分析应用题 16 的多周期处理器相比,快了 $(35.4-13.8)/13.8=1.57$ 倍。

第 8 章 系统总线

8.1 教学目标和内容安排

主要教学目标：

使学生掌握现代计算机中各主要模块之间的总线互连方式。包括总线基本概念、总线的分类、总线的组成及性能指标、总线的裁决、总线的定时方式、总线标准及其现代计算机的总线互连结构。

基本学习要求：

(1) 了解总线的作用和组成。

(2) 了解总线的不同分类。

(3) 理解总线各项性能指标的含义。

(4) 理解为什么要采用总线标准。

(5) 理解计算机如何通过总线连接各功能部件。

(6) 理解总线如何确定总线控制权。

(7) 理解总线上的操作如何定时。

(8) 了解常用的几类总线标准。

(9) 了解典型的多总线结构框架。

本章着重介绍在计算机主要部件(CPU、存储器和 I/O 接口)之间进行连接的系统总线所涉及的概念和知识体系。在此之前，学生已经对指令的执行、存储器的层次结构、CPU 设计等方面的内容有所了解，因此，在本章开头，可以从指令执行过程中涉及的部件之间的数据交换开始讲起，将总线和指令执行过程、总线和 CPU、cache、存储器、显卡等 I/O 接口联系起来，说明指令执行过程中可能需要在哪些部件之间进行数据交换，以及在每次数据交换过程中，在部件之间需要传送哪些信息，并说明传送这些信息的任务就是由总线完成的。这样，既复习了所学的概念和知识，又将相关内容串接了起来，并且，使得学生在学习具体的有关总线设计的内容之前，对总线的组成、职责和总线在计算机中的位置有较为全面的了解，从而有利于学生对相关总线设计内容的理解。

在课时有限的情况下，本章中的几种分布式裁决方式、分离事务方式的细节、总线接口单元的基本功能，以及几个具体的总线标准都可以跳过不讲，但是，这些细节内容(如 PCI 总线标准)对主要概念的理解是非常有用的，可以要求学生在课后自己阅读这些内容，其阅

读的目的不是要学生去记住具体的实现细节，而是通过对具体实现细节的了解来强化对基本原理的理解。

8.2 主要内容提要

1. 总线的概念和分类

总线是计算机系统中部件或设备之间传送信息的公共通路，包括传输介质和相应的控制逻辑。根据总线所在位置，可以分为内部总线、系统总线和通信总线 3 类。内部总线指芯片内部连接各元件的总线，如 CPU 内部总线。系统总线指在计算机的主要功能部件(CPU、主存、I/O 接口)之间传送信息的总线，由数据线、地址线和控制线组成。根据所处位置和功能的不同，系统总线又可分为处理器总线、存储器总线和 I/O 总线。通常处理器总线和存储器总线是专用总线，而 I/O 总线是标准总线，如 PCI 总线、AGP 总线、PCI-Express 总线等。通信总线指用于主机和 I/O 设备之间或计算机系统之间通信的总线，如 RS-232 串行总线、USB 串行总线、SCSI 总线等。

2. 总线带宽

指总线的最大数据传输率，即在数据传输阶段单位时间内总线上可传输的数据量。它与总线位宽、总线时钟频率和传送每个数据所需要的时钟周期数有关。

3. 总线事务类型

指在总线上进行的不同信息传输类型，如存储器读、存储器写、I/O 读、I/O 写、中断响应等。有些总线事务要求完成一连串连续单元的读写，如从存储器读一个 cache 行或写一个 cache 行到主存，这种情况下，一个总线事务能完成多个数据的读写，称为突发(Burst)传输方式。

4. 总线裁决方式

在多个主控设备都要求使用总线时，需要进行总线裁决，有集中式裁决和分布式裁决两类。

集中式裁决由专门的总线裁决器来集中确定总线使用权。集中式裁决主要有链式查询、计数器定时查询和独立请求方式。链式查询方式下，总线允许信号在设备间用菊花链串联起来，按顺序查询，接口简单；计数器定时查询方式下，通过计数值确定查询顺序，优先级比链式查询灵活，能保证公平性；独立请求方式下，每个设备有独立的总线请求线，优先级可编程设置，裁决速度快。

分布式裁决没有专门的总线裁决器，裁决逻辑分散在每个设备的总线接口中。分布式裁决主要有自举分布、冲突检测和并行竞争方式。自举分布方式下，每个设备只有在查看到所有优先级比自己高的设备没有请求时才能使用总线；冲突检测方式下，设备直接使用总线，使用过程中侦听到有其他设备也在使用总线时，则释放总线，然后在一个随机时间段后再次使用总线；并行竞争方式下，每个设备接口中有专门的仲裁逻辑，能保证送到数据线上的多个仲裁号(设备号)中只有最大的仲裁号的信息保留在数据线上，以取得总线使用权。

5. 总线定时方式

在总线上通信的两个设备必须知道对方何时传送什么信息，因此双方需要有相应的通信协议，以确定如何交换信息，这就是定时方式的确定。最基本的定时方式有同步和异步两

种，在这两种基本定时方式基础上，又派生出半同步和分离事务两种定时方式。同步方式用一个公共的时钟信号对传输过程的每个步骤进行同步控制；异步方式用异步应答(握手)信号对传输过程的每个步骤进行定时控制；半同步方式将同步和异步方式结合起来进行定时，即在时钟信号的同步控制下发出和采样异步应答信号；分离事务方式把传输过程分成两个阶段，使得从设备在准备数据时总线被释放给其他设备使用。

6. 总线标准

总线标准化以后，使得在计算机中增删设备非常容易，提高了设备的兼容性和互换性。因此，I/O 总线和通信总线大多是标准化总线。曾经流行或正在被广泛使用的 I/O 总线标准有 ISA、EISA、PCI、APG、PCI-Express 等。

7. 总线结构

随着计算机系统中相互连接的部件或设备种类的增加，用于连接部件和设备的系统总线的数量也越来越多。计算机连接结构从早期的单总线结构发展到现在的多总线结构。单总线结构中，所有主要功能部件(如 CPU、主存和各 I/O 接口模块)都挂接在一个总线上。双总线结构中，CPU、主存、I/O 接口之间分别互连，以形成不同的总线。如 CPU 和主存之间用处理器-主存总线；CPU 和 I/O 接口模块之间用 I/O 总线；CPU、主存和 IOP 之间用主存总线；各 I/O 接口模块和 IOP 之间用 I/O 总线；等等。多总线结构将不同速度的部件细分后再分级互连，总线和总线之间用桥接器相连，以形成多总线结构。如 cache 和 CPU 之间单独用局部总线相连；处理器总线和存储器总线之间加一个桥接器，分别与处理器和存储器相连；将高速 I/O 设备和低速 I/O 设备分离，分别用高速 I/O 总线和慢速 I/O 总线相连。

8.3 基本术语解释

总线(Bus)

总线是共享的信息传输介质，用于连接若干设备，由一组传输线组成，信息通过这组传输线在设备之间被传送。总线按其所在的位置，分为片内总线、系统总线和通信总线。

片内总线(Internal Bus)

片内总线指芯片内部连接各元件的总线。例如，在 CPU 芯片内部，存在连接各个寄存器、ALU、多路选择器等各元件的总线。

系统总线(System Bus)

系统总线是用来连接计算机硬件系统中若干主要部件(如 CPU、主存、I/O 模块)的总线。系统总线上传输的有数据、地址和控制信息，因此系统总线中包含 3 组传输线：数据线、地址线和控制线，有时也把它们分别称为数据总线、地址总线和控制总线。根据总线所在的位置，系统总线可分为处理器总线、存储器总线和 I/O 总线。(注：Intel 公司推出的芯片组中，对系统总线赋予了特定的含义，把 CPU 连接到北桥芯片的总线称为系统总线，也称为处理器总线，或称为前端总线(Front Side Bus，FSB)。CPU 通过前端总线连接到北桥芯片，进而通过北桥芯片和内存、显卡交换数据)

处理器—存储器总线(Processor-memory Bus)

把专门用来连接处理器和主存的总线称为处理器—存储器总线。这种总线不是通用的标准总线，而是按机器定制的专用总线。这类总线长度短，速度快。其性能指标是主板的最

重要性能指标之一。

如果在处理器和主存之间有一个中间部件(通常为桥接器),那么处理器—存储器总线被分成处理器总线和存储器总线。例如,在 Intel 公司推出的芯片组中,因为在 CPU 和主存之间有一个北桥芯片组,所以,处理器—存储器总线被分成了两个总线:把 CPU 连接到北桥芯片的总线称为处理器总线,也称为系统总线或前端总线,把北桥芯片连到主存的总线称为存储器总线。所以,CPU 通过前端总线(FSB)连接到北桥芯片,进而通过存储器总线,再连到内存;另外,从北桥通过 AGP 总线或 PCI-Express 总线可以连到显示卡,以便和显示卡交换数据。

I/O 总线(I/O Bus)

用于各种外设控制器(即 I/O 接口,如显卡、网卡等)与主机相连,通常是标准总线,如 PCI 总线、AGP 总线、PCI-Express 总线等。

通信总线(Communication Bus)

通信总线用于主机和 I/O 设备之间或计算机系统之间的通信。由于这类连接涉及各类不同设备,其距离远近、速度快慢、工作方式等差异很大,所以通信总线的种类很多。如 RS-232 串行总线、USB 串行总线、SCSI 总线等。

底板总线(Backplane Bus)

总线按连线类型可分为电缆式、主板式和底板式。通常,通信总线采用电缆式总线形式,处理器—存储器总线采用主板式总线形式,而 I/O 总线采用底板式总线形式。

底板式总线在主板上提供相应的总线插槽,各种 I/O 接口模块(如网卡、显卡等)以 I/O 插件板卡的形式插入相应的插槽,即可通过总线完成信息的交换。为使各 I/O 插件板的插座之间具有通用性,底板总线通常是标准总线,使得不同厂家的 I/O 插件板可以互连、互换。

并行总线(Parallel Bus)

并行总线的数据在数据线上同时有多位一起传送,每一位要有一根数据线,因此由多根数据线组成。其特点是同时可以传输多位数据,但数位之间必须要同步。并行总线因为数据线多,使得相互间干扰大,因而,数据传输的速度受到很大影响。

串行总线(Serial Bus)

串行总线的数据在数据线上按位进行传输,因此只需要一根或两根数据信号线,线路的成本低,适合远距离的数据传输。此外,提高时钟速度比并行连接容易得多,且几乎没有线间串扰,因而串行总线的速度可以比并行总线快得多。

信号线复用(Signal Multiplexing)

信号线复用指同一组线在不同的时刻传送不同的信号,如数据/地址线复用时,用一组线在总线事务的地址阶段传送地址信息,在数据阶段传送数据信息,这样就使得地址和数据通过同一组线进行传输。

总线宽度(Bus Width)

总线中数据线的条数被称为总线的宽度,它决定了每次能同时传输的数据信息的位数。用位表示时,也称为总线位宽;用字节表示时,其值为总线位宽除以 8。

总线时钟频率(Bus Clock Frequency)

总线中用于定时的公共时钟信号的频率就是总线的时钟频率。常以 MHz、GHz 等为单位。这里 M 和 G 都是 10 的幂次。

总线事务(Bus Transaction)

通常把在总线上一对设备之间的一次信息交换过程称为一个“总线事务”。把发出事务请求的部件称为主控设备或主设备(Master),也称起动者(Initiator),另一个部件称为从设备(Slave),也称目标(Target)。总线事务类型由其操作性质来定义。例如,一个“存储器读”事务是指将数据从主存取出到处理器,一个“存储器写”事务是指将数据写到主存。典型的总线事务类型有“存储器读”、“存储器写”、“I/O 读”、“I/O 写”、“中断响应”等。

主控设备(Master,Initiator)

发起总线请求并在获得总线使用权后能控制总线的设备。如 CPU、DMA 控制器等都可以用作主控设备。

从设备(Slave,Target)

不能发出总线请求信号,对总线没有控制能力的设备,它只能响应从主控设备发来的总线命令。例如,主存通常只能是从设备。

总线传输周期(Bus Transmission Cycle)

指总线上完成一次总线事务所用的时间,通常它由若干个总线时钟周期组成,简称为总线周期。

总线带宽(Bus Bandwidth)

总线的带宽就是总线的最大数据传输率,即总线在进行数据传输时单位时间内在总线上可传输的最大数据量。它与总线宽度、总线时钟频率等有关。用公式表示为总线带宽 $B=W\times F/N$ (式中W 为总线宽度,F 为总线时钟工作频率,N 为传输一个数据所用的总线时钟周期数)。

突发传送方式(Burst Transmission)

是一种连续的、成批数据传送方式。只需在传送开始时给出数据块的首地址,然后连续传送多个数据,后续数据的地址被默认为前面数据地址加 1,实际上这些连续的数据一般在存储阵列的同一行上。在存储器中有一个行缓冲,一次读出一行数据,后面的连续数据只要从行缓冲中源源不断地取出即可。所以,这种方式能够实现一个时钟传送一个或两个数据,在总线宽度和工作频率相同的情况下,数据传输率大大高于不支持突发传送的总线。这种方式也称为背对背(Back-to-Back)传送方式。

总线裁决(Bus Arbitration)

决定哪个总线主控设备将在下次得到总线使用权的过程被称为总线裁决。进行总线裁决有多种方案,主要分为集中式和分布式两大类。

总线裁决器(Bus Arbiter)

总线裁决器也叫总线控制器,当多个设备同时请求使用总线时,需要通过总线裁决器决定由哪个设备控制总线。

总线请求信号(Bus Request Signal)

一种控制信号,设备要求使用总线时发出,用于表明发出该请求信号的设备需要使用总线。

总线响应/允许信号(Bus Acknowledge/Bus Grant Signal)

一种控制信号,总线控制器在接受到设备的总线请求后,根据优先级选择一个设备发送总线响应信号,表示允许该设备在下一个总线周期使用总线。

集中裁决(Centralized Arbitration)

集中裁决方式将总线控制逻辑做在一个专门的总线控制器或总线裁决器中，通过将所有的总线请求集中起来利用一个特定的裁决算法进行裁决。

分布裁决(Distributed Arbitration)

分布裁决方式没有专门的总线控制器，其控制逻辑分散在各个总线部件或设备中。

同步总线(Synchronous Bus)

各部件采用统一的时钟信号进行同步，协议简单，因而速度快，接口逻辑很少。但总线上的每个部件必须在规定的时间内完成要求的动作，因此一般按最慢的部件来设计公共时钟。由于时钟偏移问题，同步总线不能很长。所以，一般同步总线用在部件之间距离短、存取速度较一致的场合。例如：CPU 片内总线、处理器—存储器总线大都采用同步方式。

异步总线(Asynchronous Bus)

异步总线中没有时钟信号线，每一步操作不靠时钟定时。为了协调异步总线在发送和接收者之间的数据传送，异步总线使用一种握手协议(应答信号)进行通信。允许各设备之间的速度有较大的差异，所以异步总线大多用于具有不同存取速度的设备之间进行通信。通常，异步方式被通信总线采用。

握手信号(Handshaking Signal)

握手协议由一系列步骤组成，在每一步中，只有当双方都同意时，发送者或接收者才会进入下一步，协议是通过一组附加的控制线来实现的。握手协议中的信号被称为握手信号或应答信号。

半同步总线(Semi-Synchronous Bus)

半同步通信总线既保留了同步通信的特点，又能采用异步应答方式连接速度相差较大的设备。通过在异步总线中引入时钟信号，其“就绪”和“完成”等应答信号都在时钟的上升沿或下降沿被采样，因而不受其他时间信号的干扰。底板式的 I/O 总线大多采用半同步方式。

分离事务协议(Split Transaction Protocol)

将一个事务过程分成两个子过程，在第一个子过程中，主控设备 A 在获得总线使用权后，将请求的事务类型(总线命令)、地址以及其他有关信息(如标识主控设备身份的编号等)发送到总线上，从设备 B 记下这些信息。主控设备发完这些信息后便立即释放总线，这样其他设备便可使用总线；在第二个子过程中，从设备 B 收到主控设备 A 发来的信息后，就按照其请求的命令进行相应的操作，当准备好主控设备所需要的数据后，从设备 B 便请求使用总线，一旦获得使用权，则从设备 B 就将主控设备 A 的编号及所需要的数据等送到总线上，这样主控设备 A 便可接收数据。这种事务分离方式的优点是：通过在不传送数据期间释放总线，使得其他申请者能使用总线，提高了总线使用效率。但是这种方式的控制相当复杂，开销也大，并且事务响应时间也会变长。

即插即用(Plug and Play)

即插即用功能是指任何扩展卡只要插入系统的插座上就能工作。这种技术通过自动配置计算机中的板卡和其他设备，将物理设备和软件(设备驱动程序)相配合，并在每个设备和它的驱动设备之间建立通信信道。

8.4 常见问题解答

1. 机器字长、编址单位、存取单位、传输单位、指令字长各指什么？它们之间有何关系？

答：在计算机内部，有指令和数据两大类信息。指令和数据都以二进制形式存放在存储器中，运行程序时，需要把指令和数据从存储器读出，再通过总线传输到CPU，然后，CPU再通过执行指令来对操作数进行相应的运算，最后把结果数据送到寄存器或存储器中。所以，在设计或使用计算机过程中，要涉及很多问题，例如，指令和数据在存储器中按什么长度存放；写入或读出时按什么长度存取；在总线上传输时同时传送多少位；数据和指令送到CPU后，在CPU的寄存器中按多少位存放；在运算器中按多少位来运算，等等。因而，出现了一系列有关信息单位和信息宽度的概念，这些概念非常重要，但比较容易混淆，需要将很多知识和概念联系在一起，才能很好地理解这些概念及其相互之间的关系。

它们的定义和关系说明如下。

(1) 机器字长是计算机的一个非常重要的指标。通常所谓32位机器或64位机器，就是指机器的字长是32位或64位。一般情况下，机器字长定义CPU中定点运算数据通路的位数。在计算机中，"字"的概念经常出现。一个"字"的宽度并不等于机器字长。"字"作为机器中所有信息宽度的计量单位，对于某个系列机来说，其字宽总是固定的。例如，在80x86系列中，一个字的宽度为16位，因此，32位是双字，64位是4个字。

(2) 编址单位就是存储单元的宽度，指存储器中具有相同地址的若干个存储元件（存储元、存储基元、记忆单元）构成的一个二进制代码的宽度，可以是8位、16位、32位等。现代大多数计算机按字节编址，即编址单位为8位，每一个字节有一个地址。由此可见，一个数据（如32位整数、32位浮点数或64位浮点数等）可能占多个存储单元，CPU要求一次从存储器读出或写入的信息也可能有多个存储单元。

(3) 存取宽度是指一次从一个由多个DRAM芯片构成的存储模块中同时读写的信息的宽度，例如，假定某个存储模块由8个4096×4096×8位的DRAM芯片按交叉编址方式构成，则该存储模块的存取宽度是64位，也即，8个芯片可同时读写，每个芯片同时读写8位，因而最多可以同时存取64位信息。按字节编址方式下，存取宽度为8个存储单元，存取单位可以是8位、16位、32位或64位等。

(4) 传输宽度就是总线宽度，也就是一次最多能在总线上传输的信息位数。对于存储器总线来说，总线上传输的信息宽度应该等于存储器的存取宽度。因此，在设计系统时，应考虑传输宽度和存取宽度的匹配，并且每个设备中的总线接口部件也要与这些宽度匹配。

(5) 指令字长指指令的位数。有定长指令字机器和不定长指令字机器。定长指令字机器中所有指令的位数是相同的，目前定长指令字大多是32位指令字。不定长指令字机器的指令有长有短，但每条指令的长度一般都是8的倍数。因此，一个指令字在存储器中存放时，可能占用多个存储单元；从存储器读出并通过总线传输时，可能分多次进行，也可能一次读多条指令。

2. 数据总线、地址总线和控制总线是分开连接在不同设备上的3种不同的总线吗?

答:不是。它们只是系统总线的3个组成部分,而不能分开来单独连接设备。系统总线用来连接计算机中若干主要部件,一般把在这些部件之间传输的信息分为数据、地址和控制3类。控制信号包括总线命令、定时信号(时钟和握手信号等)、总线请求、总线允许、中断请求和中断允许等,所以系统总线相应也就分成了3组传输线:数据线、地址线、控制线,有时习惯于把它们分别称为:数据总线、地址总线和控制总线。

3. 为什么要有总线判优控制?

答:总线是共享的信息传输介质,同时可以有很多设备连接在同一个总线上,但每一时刻总线只能完成一对设备之间的信息传送。当有多个设备同时要使用总线传输信息时,如果允许它们同时把自己的信息发到总线上,就会造成混乱,因此引入了总线判优机制,它能在多个请求使用总线的设备中选择一个,让其控制总线来传输信息,其他设备则需暂时等待并在以后的判优中逐一被选中。

4. 一台机器里面只有一个总线吗?

答:不一定。总线按其所在的位置,分为片内总线、系统总线、通信总线。系统总线是指在CPU、主存、I/O各大部件之间进行互连的总线。可以把所有大的功能部件都连接在一个总线上,也可以用几个总线分别连接不同的部件和设备。因此,有单总线结构、双总线结构、三总线结构,等等。通常,一台机器里面应该有不同层次的多个总线存在。

5. 一个总线在某一时刻可以有多对主、从设备进行通信吗?

答:不可以。在某一个总线传输周期内,总线上只能有一个主控设备控制总线,选择一个从设备与之进行通信,或对所有其他设备进行广播通信。所以,某一时刻一个总线不能有多对主、从设备进行通信。

6. 同步总线和异步总线的特点各是什么? 各自适用于什么场合?

答:同步总线的特点是各部件采用时钟信号进行同步,协议简单,因而速度快,接口逻辑很少。但总线上的每个部件必须在规定的时间内完成要求的动作,所以一般按最慢的部件来设计公共时钟。而且由于时钟偏移问题,同步总线不能很长。所以,一般同步总线用在部件之间距离短、存取速度较一致的场合。通常,CPU内部总线、处理器总线等采用同步总线。近年来,主存逐步采用同步的DRAM芯片构成,因此存储器总线也逐步采用同步总线。

异步总线采用应答方式进行通信,允许各设备之间的速度有较大的差异,因此,通常用于在具有不同存取速度的设备之间进行通信。通常连接外设或其他机器的通信总线采用异步总线。

7. 同一个总线不能既采用同步方式又采用异步方式通信,是吗?

答:不是的,半同步通信总线可以这样。这类总线既保留了同步通信的特点,又能采用异步应答方式连接速度相差较大的设备。通过在异步总线中引入时钟信号,其就绪和应答等信号都在时钟的上升沿或下降沿有效,而不受其他时间信号的干扰。通常I/O总线采用半同步方式。例如,PCI总线是一种半同步总线,它的所有事件在时钟下降沿同步,总线设备在时钟开始的上升沿采样总线信号。

8.5 单项选择题

1. 下列有关总线的叙述中，错误的是（　　）。
   A. 总线是一组共享的信息传输线
   B. 系统总线中有地址、数据和控制3组传输线
   C. 同步总线一定有时钟信号线，用于总线操作的定时
   D. 系统总线始终由CPU控制和管理

2. 假定一个同步总线的工作频率为33MHz，总线中有32位数据线，每个总线时钟传输一次数据，则该总线的最大数据传输率为（　　）。
   A. 66MB/s　　B. 132MB/s　　C. 528MB/s　　D. 1056MB/s

3. 下列有关同步总线事务的描述中，错误的是（　　）。
   A. 一个总线事务所用时间由多个总线时钟周期组成
   B. 总线事务开始时通常先把地址和读/写命令送到总线上
   C. “存储器读”总线事务中数据和地址通常不会同时送到总线上
   D. 一次总线事务只能完成一个数据交换，其位数不超过总线宽度

4. 下列有关异步总线事务的描述中，错误的是（　　）。
   A. 总线事务总是在一个主控设备和一个从设备之间进行
   B. 总线事务由多个握手过程组成，每次握手完成一个数据交换
   C. 一个总线事务中一定包括“地址”和“数据”两种信息的交换
   D. “I/O读”和“I/O写”总线事务源于CPU对I/O指令的执行

5. 系统总线中控制线的主要功能是（　　）。
   A. 提供定时信号、操作命令和各种请求/回答信号等
   B. 提供数据信息
   C. 提供时序信号
   D. 提供主存和I/O模块的回答信号

6. 下列有关同步总线的描述中，错误的是（　　）。
   A. 用一个公共时钟信号进行同步
   B. 不需要应答（握手）信号
   C. 要求挂接在总线上的各部件的存取时间较为接近
   D. 总线长度不受限制，可以很长

7. 下列关于异步总线的叙述中，错误的是（　　）。
   A. 需要应答（握手）信号
   B. 需用一个公共的时钟信号进行同步
   C. 可以实现高可靠的数据传输
   D. 挂接在总线上的各部件可以有较大的速度差异

8. 以下总线裁决控制方式中，（　　）方式对电路故障最敏感。
   A. 链式查询　　B. 计数器定时查询
   C. 独立请求　　D. 自举分布

9. 假定有 n 个设备挂接在总线上，采用链式查询方式时需要的总线请求线的条数为(　　)。

A. 1　　B. 2　　C. n　　D. $2n$

10. 假定有 n 个设备挂接在总线上，采用独立请求方式时需要的总线请求线的条数为(　　)。

A. 1　　B. 2　　C. n　　D. $2n$

11. 在计数器定时查询方式下，若每次计数都从 0 开始，则(　　)。

A. 设备号小的设备优先级高　　B. 设备号大的设备优先级高

C. 每个设备的优先级均等　　D. 每个设备的优先级随机变化

12. 在计数器定时查询方式下，若每次计数都从上次得到响应的设备随后一个设备号开始，则(　　)。

A. 设备号小的设备优先级高　　B. 设备号大的设备优先级高

C. 每个设备的优先级均等　　D. 每个设备的优先级随机变化

13. 增加同步总线带宽的手段有很多，但以下(　　)不能提高总线带宽。

A. 采用信号线复用技术　　B. 增加总线宽度

C. 采用突发(Burst)传送方式　　D. 提高总线时钟频率

14. 以下有关总线标准的叙述中，错误的是(　　)。

A. 引入总线标准便于设备互换和新设备的添加

B. 主板上的处理器总线和存储器总线通常是专用总线

C. I/O 总线通常是标准总线，所以 PCI 总线是标准总线

D. 串行总线的数据传输率一定比并行总线的数据传输率低

15. 下列选项中的英文缩写均为总线标准的是(　　)。

A. PCI、CRT、USB、EISA　　B. ISA、CPI、VESA、EISA

C. ISA、SCSI、RAM、MIPS　　D. ISA、EISA、PCI、PCI-Express

16. 以下给出的总线标准中，不属于 I/O 总线标准的是(　　)。

A. PCI　　B. PCI-Express　　C. SCSI　　D. AGP

17. 以下有关多总线结构系统的叙述中，错误的是(　　)。

A. 通常越靠近 CPU 的总线传输速率越高

B. 通常在总线和总线之间用桥接器连接

C. CPU 总线和存储器总线都比 I/O 总线快

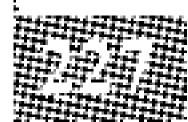
D. 系统中的多个总线不可能同时传输信息

【参考答案】

1. D	2. B	3. D	4. B	5. A	6. D	7. B
8. A	9. A	10. C	11. A	12. C	13. A	14. D
15. D	16. C	17. D				

8.6　分析应用题

1. 假设一个 32 位的处理器连接了一个 32 位宽的处理器总线，总线的时钟频率为 400MHz，支持多种总线事务类型。其中，最短的总线事务类型是存储器读事务，需要 4 个



时钟周期完成，第1个时钟周期送地址和读命令，第4个时钟周期取数；最长的总线事务类型是突发传送8次数据，需要11个时钟周期完成，第1个时钟周期送地址和读命令，第4个时钟周期开始连续传送8个数据，每个时钟周期传送一次。请回答下列问题：

（1）该总线是同步总线还是异步总线，为什么？

（2）该总线的最大数据传输率为多少？

（3）若处理器一直持续发起最短总线事务类型，则此时总线的数据传输率是多少？

（4）若处理器一直持续发起最长总线事务类型，则此时总线的数据传输率是多少？

（5）若将处理器总线的总线宽度扩展为64位，则该总线的最大数据传输率提高到多少？

（6）若将处理器总线的总线时钟频率提高到800MHz，则该总线的最大数据传输率提高到多少？

（7）加倍总线宽度和加倍总线时钟频率相比，哪种更好？

【分析解答】

（1）该总线是同步总线，因为所有总线操作都在总线时钟的控制下进行。

（2）总线最大数据传输率就是总线带宽，表示在总线上传输数据时单位时间内传输的最大数据量，它由总线宽度W、总线时钟频率F和一个时钟周期内传输的数据个数M确定。其值等于W×F×M。因此该总线的最大数据传输率为32b×400MHz×1＝12.8Gb/s＝1.6GB/s。

（3）持续进行最短总线事务类型时，总线数据传输率为4B×400MHz/4＝400MB/s。

（4）持续进行最长总线事务类型时，总线数据传输率为4B×8×400MHz/11＝1164MB/s。

（5）若总线宽度扩展一倍，则总线最大数据传输率提高一倍，即为3.2GB/s。

（6）若总线时钟频率提高一倍，则总线最大数据传输率提高一倍，即为3.2GB/s。

（7）加倍总线宽度和加倍总线时钟频率的措施对于总线速度来说，效果是一样的。

2. VAX SBI总线采用分布式的自举裁决方案，总线上每个设备有唯一的优先级，而且被分配一根独立的总线请求线REQ，SBI有16根这样的请求线（REQ0，…，REQ15），其中REQ0优先级最高，请问：最多可有多少个设备连到这样的总线上？为什么？

【分析解答】

最多可连接16个设备。因为在分布式自举裁决方式的总线中，除优先级最低的设备外，每个设备都使用一根信号线发出总线请求信号，以被优先级比它低的设备查看；而优先级最低的那个设备无须送出总线请求信号。此外，还需要一根总线请求信号线用于设置“总线忙”信号，所以，16根总线请求线中，有15根分别用于15个不同设备的总线请求，故最多可挂接15＋1＝16个设备。

3. 试设计一个采用固定优先级的具有4个输入的集中式独立请求裁决器。

【分析解答】

假设BR_0～BR_3为4个总线请求线，优先级由高到低，BG_0～BG_3为对应的4个总线允许线，则固定优先级的并行判优电路的逻辑方程如下。

$BG_0=BR_0$；

$BG_1=BR_1\overline{BR_0}$；

$BG_2 = BR_2\overline{BR_1}\ \overline{BR_0}$；

$BG_3 = BR_3\overline{BR_2}\ \overline{BR_1}\ \overline{BR_0}$。

根据上述逻辑表达式，不难实现一个独立请求裁决器。

4. 假设某存储器总线采用同步通信方式，时钟频率为50MHz，每个总线事务以突发方式传输8个字，以支持块长为8个字的cache行读和cache行写，每字4字节。对于读操作，访问顺序是1个时钟周期接收地址，3个时钟周期等待存储器读数，8个时钟周期用于传输8个字。对于写操作，访问顺序是1个时钟周期接收地址，2个时钟周期延迟，8个时钟周期用于传输8个字，3个时钟周期恢复和写入纠错码。对于以下访问模式，求出该存储器读/写时在存储器总线上的数据传输率。

(1) 全部访问为连续的读操作。

(2) 全部访问为连续的写操作。

(3) 65%的访问为读操作，35%的访问为写操作。

【分析解答】

(1) 读取8个字用了1+3+8=12个时钟周期，故数据传输率为8×4B/(12×1/50MHz)=133.3MB/s。

(2) 写入8个字用了1+2+8+3=14个时钟周期，故数据传输率为8×4B/(14×1/50MHz)=114.3MB/s。

(3) 可用两种方式估算。若用数据传输率加权平均，则为133.3MB/s×65%+114.3MB/s×35%=126.7MB/s；若用时钟周期数的加权平均，则为8×4B/((12×65%+14×35%)×1/50MHz)=126.0 MB/s。

5. 在一个字长为32位的计算机系统中，假定存储器分别连接以下两种不同的同步总线。

总线1是64位数据和地址复用的总线。能在一个时钟周期中传输一个64位的数据或地址，支持最多连续8个字的存储器读和存储器写总线事务。任何一次读写操作总是先用一个时钟周期传送地址，然后有两个时钟周期的延迟等待，从第4时钟周期开始，存储器准备好数据，总线以每个时钟周期两个字的速度传送，最多传送8个字。

总线2是分离的32位地址和32位数据的总线，支持最多连续8个字的存储器读和存储器写总线事务。读操作过程为：一个时钟周期传送地址，两个时钟周期延迟等待，从第4时钟周期开始，存储器准备好数据，总线以每时钟一个字的速度传输最多8个字。对于写操作，在第一个时钟周期内，第一个数据字与地址一起传输，经过两个时钟周期的延迟等待后，第一个字写入存储器，并在后面7个时钟周期中，以每个时钟一个字的速度最多传输7个余下的数据字。

假定这两种总线的时钟频率都为100MHz，请回答下列问题。

(1) 两种总线的最大数据传输率(总线带宽)分别为多少？

(2) 连续进行单个字的存储器读总线事务时，两种总线的数据传输率分别是多少？

(3) 连续进行单个字的存储器写总线事务时，两种总线的数据传输率分别是多少？

(4) 每次传输8个字的数据块，60%是读操作总线事务，40%是写操作总线事务，两种总线的数据传输率分别是多少？

(5) 通过对以上各种数据的分析对比，给出相应的结论。

【分析解答】

(1) 总线 1 在传送数据时以每个时钟周期两个字的速度进行，所以它的最大数据传输率为 32b×2×100MHz=6400Mb/s=800MB/s。总线 2 在传送数据时以每个时钟周期一个字的速度进行，所以它的最大数据传输率为 32b×100MHz=3200Mb/s=400MB/s。

(2) 总线 1 虽然每个时钟周期可传送两个字，但在单字传输总线事务中每次只需要传送一个字，每个总线事务占 1+2+1=4 个时钟周期，因此连续进行单个字的存储器读总线事务时，总线 1 的数据传输率为 4B×100MHz/4=100MB/s。总线 2 每个时钟周期读一个字，一个单字存储器读总线事务占 1+2+1=4 个时钟周期，因此连续进行单个字的存储器读总线事务时，总线 2 的数据传输率也为 100MB/s。

(3) 总线 1 的单字存储器写总线事务和单字存储器读总线事务的情况一样，因此，连续进行单个字的存储器写总线事务时，数据传输率也是 100MB/s。总线 2 的单字存储器写总线事务占 1+2=3 个时钟周期，因此连续进行单个字的存储器写总线事务时，其数据传输率为 4B×100MHz/3=133.3MB/s。

(4) 通过总线 1 进行存储器读和写 8 个字所用时间都为 1+2+8/2=7 个时钟周期，所以在连续进行多个 8 字突发传送总线事务时，总线 1 的数据传输率为 8×4B×100MHz/7=457MB/s。总线 2 的存储器读事务和存储器写事务所用时间不等，突发读 8 个字所用的时间为 1+2+8=11 个时钟周期，突发写 8 个字所用的时间为 1+2+7=10 个时钟周期，因此，当 60%是读操作总线事务，40%是写操作总线事务时，总线 2 的数据传输率为 8×4B×(100MHz/11)×60%+8×4B×(100MHz/10)×40%=303MB/s。

(5) 总线 1 和总线 2 的数据/地址线都是 64 位，总线 1 采用数据/地址线复用，总线 2 采用分离的数据线和地址线。以下是对两种总线在各种情况下的分析以及得出的结论。

根据(1)中对两种总线最大数据传输率的计算，可知：采用数据/地址线复用技术，能够得到更大的峰值数据传输率。因为，一旦进入到数据传输阶段，用 64 位数据线传输数据肯定比用 32 位数据线传输数据要快一倍。

根据(2)和(3)中对两种总线在单字传输情况下数据传输率的计算，可知：采用数据/地址线分离技术，可以得到更大的单字传输数据速率。因为，单字传输时，数据/地址线复用时得到的两倍宽度的数据线只能传送一个字，同时因为信号线复用而不能将数据和地址同时送出，使得写事务所用时间延长，因而，采用数据/地址线复用技术的情况下，得到的单字传输数据速率更低。

根据(4)中对突发传送 8 个数据时数据传输率的计算，可知：采用数据/地址线复用技术，能够得到更大的突发数据传输率。因为，在突发传送事务中需要连续传送多个数据，此时，64 位数据线肯定比 32 位数据线传送得快。

6. 假定主存和 CPU 之间连接的同步总线具有以下特性：支持 4 字和 16 字两种长度的数据传送，字长为 32 位，总线时钟频率为 200MHz，总线宽度为 64 位，每个 64 位数据的传送需要一个时钟周期，向主存发送一个地址需要一个时钟周期，每两个总线事务之间有两个空闲时钟周期。若访问主存时最初 4 个字的存取时间为 200ns，随后每存取一个 4 字的时间是 20ns，则在 4 字和 16 字两种传输方式下，该总线上传输 256 个字时的数据传输率分别是多少？你能从计算结果中得到什么结论？

【分析解答】

总线时钟频率为 200MHz,因而总线时钟周期为 1/200MHz=5ns。

对于 4 字传送方式,每个总线事务由一个地址传送跟一个 4 字的数据块传送组成。即每个总线事务传送一个 4 字的数据块。每个数据块的传送过程如图 8.1 所示。



图 8.1 题 6 中 4 字传送方式下的数据传送过程

首先,CPU 发送一个首地址到主存,一个时钟周期后,主存读开始的 4 个字,用了 200ns/5ns=40 个时钟周期,然后在总线上传输 4 个字,所用的时钟周期数为 4×32/64=2。在下次总线事务开始之前,最后有两个空闲时钟周期。所以一次总线事务总共需要 1+40+2+2=45 个时钟周期,256 个字需 256/4=64 个事务,因而整个传送需 45×64=2880 个时钟周期,得到总延时为 2880×5ns=14400ns。每秒钟进行的总线事务数为 $64/(14400\times10^{-9})$=4.44M。总线的数据传输率为(256×4B)/14400ns=71.11MB/s。

对于 16 字传送方式,每个总线事务由一个地址传送跟一个 16 字的数据块传送组成。即每个总线事务传送一个 16 字的数据块。每个数据块的传送过程如图 8.2 所示。

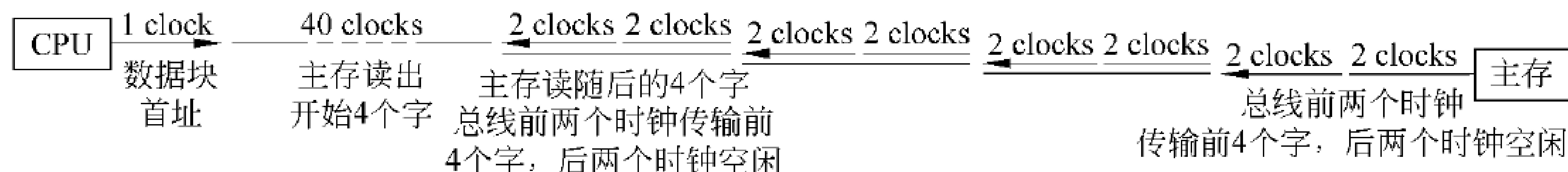


图 8.2 题 6 中 16 字传送方式下的数据传送过程

从图 8.2 可以看出,一次总线事务的时钟周期数为 1+40+4×(2+2)=57。256 个字需 256/16=16 个事务,因此整个传送需 57×16=912 个时钟周期。因而总延时为 912×5ns=4560ns,仅约 4 字传送的 1/3。每秒钟总线事务数为 $16/(4560\times10^{-9})$=3.51M/s。总线数据传输率为(256×4B)/4560ns=224.56MB/s。与 4 字传送相比,是它的 3.6 倍。

由此可见,在一次总线事务中传送的数据块越大,则数据传输率越高。

7. 在题 6 所述的系统中,假定访问主存时最初 4 个字的读取时间为 148ns,随后每读一个 4 字的时间为 26ns,则在 4 字和 16 字两种传输方式下,CPU 从主存读出 256 个字时,该总线上的数据传输率分别是多少?和上题计算结果进行比较分析,并给出相应的结论。

【分析解答】

因为最初 4 个字的读取时间从 200ns 变为 148ns,所以主存读开始的 4 个字只用了 148ns/5ns=29.6 个时钟周期,当主存存取时间不是总线时钟周期的整数倍时,主存会先准备好数据,等待下一个总线时钟周期到来后,开始在总线上传送数据。因此,开始的 4 字的读取时间实际上相当于 30 个时钟周期的时间。

4 字传输方式下,每个数据块的传送过程如图 8.3 所示。

从图 8.3 中可以看出,一次总线事务总共需要 1+30+2+2=35 个时钟周期,256 个字需 256/4=64 个事务,因而整个传送需 35×64=2240 个时钟周期,总线的数据传输率为(256×4B)/(2240×5ns)=91.43MB/s。



图 8.3　题 7 中 4 字传送方式下的数据传送过程

因为从第二个 4 字开始，每读一个 4 字的时间为 26ns，相当于 26ns/5ns＝5.2 个总线时钟周期的时间。在 16 字传输方式下，每个数据块的传送过程如图 8.4 所示。

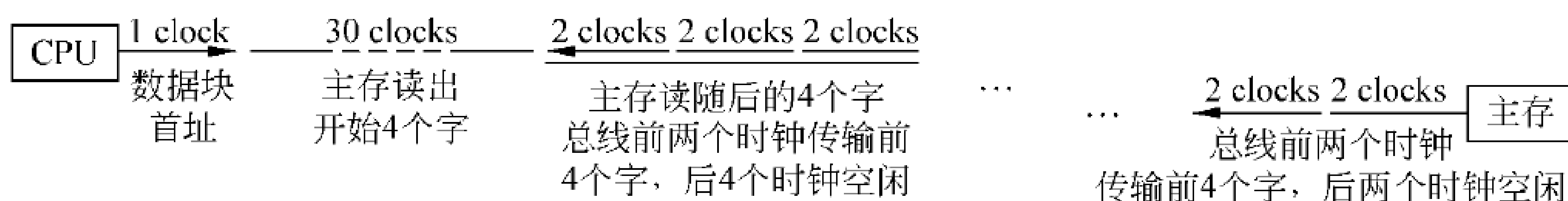


图 8.4　题 7 中 16 字传送方式下的数据传送过程

从图 8.4 中可以看出，一次总线事务总共需要 1＋30＋3×6＋2＋2＝53 个时钟周期，256 字需 256/16＝16 个事务，因而整个传送需 53×16＝848 个时钟周期，总线的数据传输率为(256×4B)/(848×5ns)＝241.51MB/s。

与第 6 题相比，4 字传输方式下，速度提高了(91.43－71.11)/71.11×100％＝28.6％；16 字传输方式下，速度只提高了(241.51－224.56)/224.56×100％＝7.5％。由此可知，对于小数据块的传输，主存首次读取的速度更重要；而对于大数据块的传输，则首次读取速度和随后的读取速度都很重要。

8. 假定主存和 CPU 之间连接的同步总线具有以下特性：采用数据和地址线分离方式，总线宽度和机器字长都为 32 位，总线时钟频率为 200MHz，每个数据或地址传送需一个时钟周期，每个总线事务之间有两个空闲时钟周期。对于写操作，主存最初 4 个字的写入时间为 200ns，随后每写入一个 4 字的时间是 20ns，则写入 256 个字到主存时，在 4 字和 16 字两种传输方式下该总线的数据传输率分别是多少？

【分析解答】

每个时钟周期为 1/200MHz＝5ns。在 4 字传输方式下，每个数据块的写入过程如图 8.5 所示。



图 8.5　题 8 中 4 字传送方式下的数据传送过程

首先，CPU 发送首地址和第一个字到主存需要 1 个时钟周期，接着发送 3 个字到主存需要 3 个时钟周期，当主存接收到地址和 4 个字后，开始将一个 4 字写入主存，需要 200ns/5ns＝40 个时钟周期。在下一次总线事务开始之前，最后有两个空闲时钟周期。所以一次总线事务总共需要 1＋3＋40＋2＝46 个时钟周期，256 个字需 256/4＝64 个事务，因而整个传送需 46×64＝2944 个时钟周期，总延时为 2944×5ns＝14720ns。总线数据传输率为(256×4B)/14720ns＝69.57MB/s。

在 16 字传输方式下，每个数据块的传送过程如图 8.6 所示。

首先，与 4 字传输方式一样，CPU 在 4 个时钟周期内，发送首地址和第一个 4 字到主

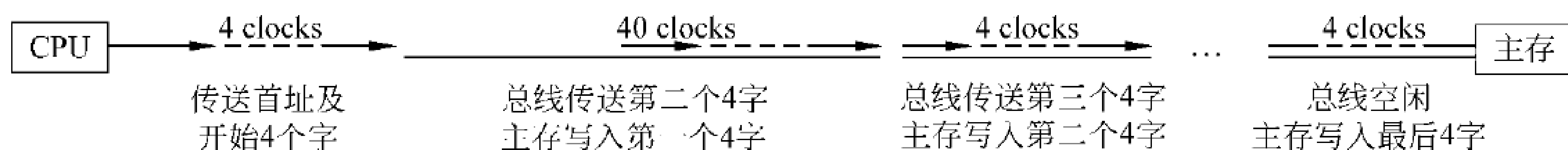


图 8.6 题 8 中 16 字传送方式下的数据传送过程

存，当主存接收到首地址和 4 字后，开始将第一个 4 字写入主存，需要 200ns/5ns＝40 个时钟周期，在此期间，总线用其中最后 4 个时钟周期传送第二个 4 字。由图 8.6 可知，一次总线事务总共需 56 个时钟周期，256 个字需 256/16＝16 个事务，因而整个传送需 56×16＝896 个时钟周期，得到总延时为 896×5ns＝4480ns。每秒钟进行的总线事务数为 $16/(4480\times10^{-9})$＝3.57M。总线的数据传输率为(256×4B)/4480ns＝228.57MB/s。

9. 某通信总线采用异步通信方式，支持突发传输，总线宽度为 32 位，数据线和地址线复用，主、从设备各有一个“就绪”信号，采用全互锁方式传送地址和数据信息，主、从设备之间每次握手最少需要 40ns。总线上的主控设备和从设备都有 1 字宽的总线接口。若每一次 I/O 请求都要求突发传输 16 个字，每个字为 32 位，则用该总线连接以下各种不同的从设备时，该总线上连续进行 I/O 请求事务时的数据传输率至多各是多少？

(1) 从设备 1：准备第一个字的时间最快为 200ns，以后每个字的准备时间最快为 20ns。

(2) 从设备 2：准备第一个字的时间最快为 200ns，以后每个字的准备时间最快为 164ns。

【分析解答】

因为采用全互锁方式传送地址和数据信息，因此，每个地址和数据信息的传送需要 3 次握手。

(1) 对于从设备 1，其 I/O 请求事务过程如图 8.7 所示。

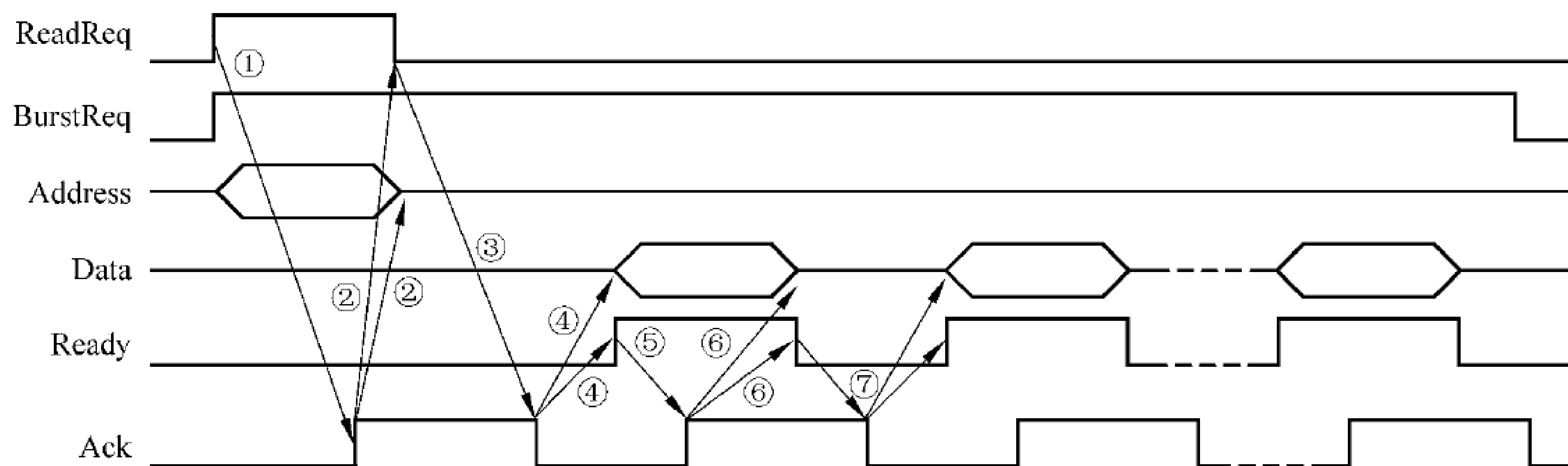


图 8.7 题 9 中异步通信方式下数据传送过程

① 主设备送出地址信息和事务类型信息(I/O 请求命令 ReadReq)，并使 BurstReq 有效，表示是突发传送方式；②从设备送出回答信号 Ack，并对地址译码，开始准备第一个字，同时，主设备接收到回答信号 Ack 后，使 ReadReq 信号无效，并撤销地址信息；③从设备跟着使 Ack 信号无效，继续准备数据；④从设备准备好第一个字后，送数据到数据线，并发出就绪(Ready)信号，主设备接收到 Ready 信号后开始读取数据；⑤主设备发出 Ack 信号，表示数据已成功读取；⑥从设备发现 Ack 信号后得知数据传送完成，因而撤销 Ready 信号并撤销数据信息；⑦主设备发现 Ready 撤销后，跟着撤销 Ack 信号。

每次数据交换的全互锁过程需要 3 次握手时间，并且在相邻两次数据交换之间需要一次握手时间，因而一次数据交换共需要 4 次握手时间，故每次交换过程所用时间为 max(4×40,20)＝160ns。每次在总线上交换第 i(1≤i<16)个字时，从设备准备第 i+1 个字，最后在总线交换第 16 个字时，从设备空闲，因而最后一个数据交换所用时间为 3×40＝120ns。最后一次交换结束后，BurstReq 信号无效，表示突发传送结束。综上所述，一次 I/O 请求传送 16 个字的时间最快为 40＋max(3×40,200)＋15×160＋120＝2760ns，因此，连续进行 I/O 请求事务时的数据传输率至多是16×4B/2760ns＝23.19MB/s。

(2) 对于从设备 2，其 I/O 请求事务过程与从设备 1 类似，只是每次准备数据所需时间不同。在总线上通过全互锁方式传输第 i(1≤i<16)个字时，从设备准备第 i+1 个字，因此前 15 个数据交换过程所用时间为 max(4×40,164)＝164ns，最后一个字的传送需 3×40＝120ns。综上所述，对于从设备 2，一次 I/O 请求传送 16 个字的时间最快为 40＋max(3×40,200)＋15×164＋120＝2820ns，因此，连续进行 I/O 请求事务时的数据传输率至多是 16×4B/2820ns＝22.70MB/s。

10. 某 I/O 总线采用半同步通信方式，支持突发传输，总线宽度为 32 位，数据线和地址线复用，总线时钟频率为 200MHz，主、从设备各有一个"就绪"信号，采用非互锁方式传送数据信息。总线上的主控设备和从设备都有 1 字宽的总线接口，每个总线事务之间有两个空闲时钟周期。若每一次 I/O 请求要求传输 16 个字，每个字为 32 位，则用该总线连接以下各种不同的从设备时，该总线上连续进行 I/O 请求事务时的数据传输率至多各是多少？

(1) 从设备 1：准备第一个字的时间最快为 200ns，以后每个字的准备时间最快为 20ns。

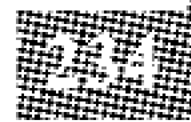
(2) 从设备 2：准备第一个字的时间最快为 202ns，以后每个字的准备时间最快为 24ns。

【分析解答】

总线采用非互锁的半同步方式传送数据信息，说明主、从设备之间只要开始时握手一次，并且，每次信息交换的时间都是时钟周期的整数倍。当主设备没有准备好接收数据时，它使自己的"就绪"信号无效，告诉从设备将当前数据继续在数据线上放置一个时钟周期；当从设备没有准备好数据时，它使自己的"就绪"信号无效，告诉主设备当前时钟没有有效数据在数据线上；总线的时钟频率为 200MHz，因此总线时钟周期为 1/200MHz＝5ns。

(1) 对于从设备 1，其 I/O 请求事务过程如下。①主设备用 1 个时钟周期送出地址信息和事务类型信息(I/O 请求命令)，并在结束时送出"就绪"信号，告知从设备自己已准备好接收数据。②从设备对地址译码并准备第一个字，用了 200ns，相当于 200/5＝40 个时钟周期。结束时，从设备送出"就绪"信号，告知主设备自己已经准备好数据。③最好的情况下，以后每隔 20ns(4 个时钟周期)从设备准备好数据，也就是说，每隔 4 个时钟周期主设备可以取到 1 个字，后续一共有 15 个字，因此用了 4×15＝60 个时钟周期。④最后两个时钟空闲。综上所述，一次 I/O 请求传送 16 个字的时间最快为(1＋40＋60＋2)×5ns＝515ns，因此，连续进行 I/O 请求事务时的数据传输率至多是 16×4B/515ns＝124.27MB/s。

(2) 对于从设备 2，其 I/O 请求事务过程如下。①主设备用 1 个时钟周期送出地址信息和事务类型信息(I/O 请求命令)，并在结束时送出"就绪"信号，告知从设备自己已准备好接收数据。②从设备对地址译码并准备第一个字，因为 202ns/5ns＝40.4，相当于 41 个时钟



周期。结束时，从设备送出“就绪”信号，告知主设备自己已经准备好数据。③最好的情况下，以后每隔24ns(相当于5个时钟周期)从设备准备好数据，也就是说，每隔5个时钟周期主设备取到1个字，后续一共有15个字，因此用了5×15＝75个时钟周期。④最后两个时钟空闲。综上所述，一次I/O请求传送16个字的时间最快为(1＋41＋75＋2)×5ns＝595ns，因此，连续进行I/O请求事务时的数据传输率至多是16×4B/595ns＝107.56MB/s。

第 9 章 输入输出组织

9.1 教学目标和内容安排

主要教学目标：

使学生了解输入/输出系统涉及的软硬件概念和知识体系，为学习操作系统中的存储管理和设备管理等内容打下坚实基础。本章主要内容包括输入/输出系统的组成、I/O对系统性能的影响、I/O设备的种类和特性、磁盘存储器的主要性能指标、I/O接口的职能和分类、I/O设备和主机的连接方式、程序查询I/O方式、中断I/O方式和DMA方式等。

基本学习要求：

(1) 了解常用外部设备的基本原理。

(2) 了解磁盘存储器的主要技术指标。

(3) 了解磁盘存储器的记录格式。

(4) 了解冗余磁盘阵列(RAID)的基本原理。

(5) 了解I/O接口的功能和基本结构。

(6) 理解I/O接口和I/O端口的差别。

(7) 了解各种I/O接口类型的特点。

(8) 理解I/O端口的编址方式。

(9) 了解各种I/O传送控制方式的特点和适用场合。

(10) 了解程序直接控制(查询)方式的特点和工作流程。

(11) 了解中断I/O方式的特点和工作过程。

(12) 了解外设向CPU提出中断请求的中断源类型。

(13) 了解CPU查询中断请求的过程。

(14) 理解CPU响应中断的3个条件。

(15) 了解CPU响应中断请求的过程。

(16) 掌握中断响应隐指令的实现方法。

(17) 掌握调出中断服务程序的方法(软件查询或形成向量地址)。

(18) 了解中断服务程序(中断处理过程)的结构框架。

(19) 理解断点保护和现场保护之间的不同。

(20) 理解为何需要进行断点保护和恢复并了解其实现方法。

(21) 了解中断允许触发器的作用以及应在何时开/关中断。

(22) 理解中断服务程序调用和子程序调用的差别。

(23) 理解中断接口电路(中断控制器)的结构。

(24) 理解多重中断和中断屏蔽的概念。

(25) 了解获取中断服务程序入口地址的过程。

(26) 理解中断返回和子程序返回的差别。

(27) 理解为什么要在中断处理开始时保护现场、结束时恢复现场。

(28) 了解 DMA 方式适用的场合和特点。

(29) 了解 DMA 接口的组成结构。

(30) 掌握 DMA 传送过程。

(31) 理解中断方式和 DMA 方式的差别。

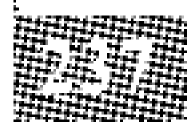
输入输出组织主要指用于控制外设与主存、外设与 CPU 之间进行数据交换的软、硬件系统。实现输入输出功能的关键是要解决以下一系列的问题：如何在 CPU、主存和外设之间建立一个高效的信息传输“通路”；怎样将用户的 I/O 请求转换成对设备的控制命令；如何使 CPU 方便地寻找到要访问的外设；I/O 硬件和 I/O 软件如何协调完成主机和外设之间的数据传送等。因此，本章的内容应围绕这些问题展开讲解。

本章的内容主要包括常用输入/输出设备、外部存储设备、I/O 接口和 I/O 数据传输控制方式这 4 个方面。

对于常用输入/输出设备，因为设备的工作原理不属于主干内容，而且不同设备的工作原理差别较大，所以，在课时有限的情况下，关于键盘、鼠标器、打印机和显示器等的工作原理可以跳过不讲。但是，如果学生对 I/O 设备的功能和结构一点不了解，那么，要理解后续的如 I/O 接口等内容就比较困难，因此，可以简单讲解 I/O 设备的通用结构，以便让学生明白：设备与计算机主机之间有数据交换，如键盘和鼠标器的输入信息、打印机和显示器的输出信息等；计算机主机会向设备发送控制信息，如打印机的初始化、选通、自动走纸等命令；设备会向计算机主机回送状态信息，如打印机忙、缺纸、联机等状态信息。

对于外部存储设备，因为它是存储器分层体系结构中的一个重要组成部分，所以它属于主干内容，特别是硬磁盘存储器，它与操作系统中的存储管理、设备管理和文件系统等都有很大的关联，因此，本课程中应该把硬磁盘存储器讲清楚，包括：性能指标、读写原理、硬磁盘驱动器的内部结构、硬磁盘控制器的结构和功能、磁道和扇区记录格式等。关于冗余磁盘阵列(RAID)，因为其应用的广泛性，作为计算机专业的学生必须对其有一定的了解，在课时有限的情况下，可以把 RAID 技术的基本思想、基本实现原理简单介绍一下。对于 U 盘、移动硬盘、固态硬盘、磁带存储器、光盘存储器等，在课时有限时，也可以不展开来细讲，而是主要介绍基本原理和主要使用场合。

对于 I/O 接口，因为不同设备对应的 I/O 接口的结构相差很大，而且各种接口的类型也很多，有限的课时无法讲清楚所有 I/O 接口的实现细节，所以，本章应主要关注 I/O 接口的通用结构和一般功能。这部分内容中，有关操作系统对 I/O 的支持，上课时可以跳过不讲，但是，如果学生课后能够很好地阅读这部分内容，则对于学生理解用户程序和操作系统之间的关系、操作系统和硬件之间的关系都有好处，从而能更好地理解计算机的层次化结构。另外，对于同步串行通信协议、I/O 接口举例(如 Intel 8255A 并行接口、SCSI 总线式并



行接口)的内容也都可以跳过不讲,而作为学生课后阅读材料,让学生了解一些实现细节,来更好地理解基本原理。

对于I/O传送控制方式,主要要求对程序直接控制、中断和DMA三种基本I/O方式有所掌握。有关DMA对存储系统带来的影响,可以不展开来介绍,而是提出问题以引起学生的思考。对于通道和输入/输出处理机方式,因为不是目前单处理器计算机系统所用的主流方式,所以细节部分可以跳过不讲,而是简单说明一下基本原理即可。

9.2 主要内容提要

1. 外部设备及其与主机的互连

输入设备、输出设备和外存储器统称为外部设备,简称外设。像键盘、鼠标、针式打印机等设备每次按单个数据为单位进行交换,属于字符型设备;磁盘、光盘、扫描仪等设备一旦被启动后,每次都会交换一块数据,因此,属于成块传送设备。

所有外设通过相应的电缆(通信总线)连到I/O接口电路上,I/O接口电路再通过I/O总线连到主板上,最终通过存储器总线和处理器总线与主存和CPU相连。

2. 常用输入、输出设备

常用的输入设备有键盘、鼠标、扫描仪等。键盘通过串行方式向主机输入信息,通常所用的键盘为非编码键盘,从键盘送到主机侧键盘接口电路中的是按键的扫描码,即位置码。鼠标器也是通过串行方式向主机传送信息,输入的是鼠标移动的位置信息。

常用的输出设备有打印机和显示器等。打印机有针式、激光和喷墨3类,它们各自适用的场合不同,所用的打印技术也不一样。激光打印机比较复杂一些,它由打印控制器和打印部件两部分组成,分别用于打印控制和打印。打印控制器中包含功能较强的专门用于打印控制的处理器、缓冲存储器和其他辅助电路,可实现对打印语言的解释、页面内容的格式化和光栅化(转换为点阵数据)等处理。显示器有CRT、液晶(LCD)和等离子显示器等,目前使用较多的是液晶显示器。显示器显示的信息是离散的像素点,显示器的主要参数有分辨率、行频、帧频(刷新频率)等。屏幕上的点的颜色用R、G、B三个分量来表示,其位数之和就是颜色深度,每个点的颜色值存放在显示存储器(VRAM,显存)中。在主机和显示器之间有一个显示控制器,可集成在主板上,也可以以显卡的形式插在I/O总线槽中。显存通常集成在显卡中,为了快速处理3D图形,通常在显卡中配置专门用作图形处理的绘图处理器(GPU),此时,显存不仅用来存放屏幕位图信息,更主要的是用来存放GPU芯片处理过或者即将处理的像素数据和渲染数据。

3. 常用外部存储器

常用外部存储器有磁盘、闪存盘、磁带库和光盘库。磁盘主要用作主存的辅助存储器,存放计算机系统的所有信息;闪存盘以U盘、外接硬盘或内装的固态硬盘形式出现;磁带库和光盘库主要用作备份数据的后备存储器。

磁盘和磁带都是磁表面存储器,其基本存储原理一样。磁盘存储器的主要技术指标如下。

(1) 记录密度:道密度指单位长度上的磁道数;位密度为磁道中单位长度上的位数。

(2) 平均存取时间:平均寻道时间和平均等待时间之和(数据传输时间相对较小,可忽

略不计)。平均寻道时间指移动磁头到所读写磁道的平均时间;平均等待时间指要读写的扇区旋转到磁头下方所需要的平均时间,等于磁盘旋转一圈所花时间的一半。

(3) 数据传输率: 分为内部数据传输率和外部数据传输率。内部数据传输率与磁盘转速有关,指寻道和旋转等待后,单位时间内从存储介质上读出或写入的二进制信息量。外部数据传输率与磁盘转速无关,指磁盘接口(磁盘控制器)和磁盘缓存之间进行数据交换的数据传输率。外部数据传输速率比内部数据传输速率高得多。

4. 冗余磁盘阵列 RAID

RAID 技术的主要实现思想是通过多个物理盘的交叉存储和访问,提高访问速度,并增加存储容量;同时,通过在盘上增加校验信息,以提高磁盘的可靠性。根据交叉存储块大小的不同,可以有小条带方式和大条区方式。对于小条带分布方式,因为一个 I/O 请求分布在所有物理盘中,而所有盘都可以并行访问,所以其数据传输率比单个盘的情况要高很多倍,因而适用于流媒体播放系统;对于大条区分布方式,因为每个交叉存储的数据块较大,使得小数据量的 I/O 请求只访问一个物理盘,多个物理盘可以响应多个 I/O 请求,所以其 I/O 响应速度较快,适用于银行、证券等事务处理系统。

RAID 级别有 RAID0～RAID7,并派生出了 RAID10、RAID30 和 RAID50 等。目前广泛使用的是 RAID0、RAID1 和 RAID5 等。

5. I/O 接口的职能、结构和类型

I/O 接口是用于连接主机和外设并通过接受主机命令来对外设进行控制的部件的总称。例如,显卡、网卡、打印控制器、磁盘控制器等都属于 I/O 接口,有时也称为 I/O 模块。

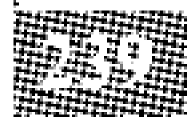
不同设备对应的 I/O 接口的功能不完全相同,其逻辑结构也不一样。但是,所有 I/O 接口的基本结构和职能是类似的。I/O 接口中,有用于存放输入/输出数据的数据缓冲器、用于记录设备或接口状态的状态寄存器、用于存放控制信息的命令(控制)寄存器等,这些寄存器分别称为数据端口、状态端口和命令(控制)端口。I/O 接口在主机一侧,通过 I/O 总线与主机相连,在外设一侧通过通信总线(电缆)与外设相连。通常 I/O 总线和通信总线的数据宽度不同,因此,在主机侧和外设侧的数据宽度不一样,因而在 I/O 接口中需要有进行数据格式转换的逻辑电路,此外,还需在主机侧和外设侧分别有相应的总线接口逻辑,以支持与 I/O 总线和通信总线的连接。

I/O 接口的类型多种多样。按设备侧能同时传输的位数来分,有并行接口和串行接口;按是否可以编程控制来分,有可编程接口和不可编程接口;按是否支持标准的通信总线来分,有通用接口和专用接口;按 I/O 方式来分,有无条件查询接口、条件查询接口、中断控制器接口、DMA 控制器接口;按连接方式来分,有点对点接口和多点总线式接口。

6. I/O 端口及其编址

I/O 端口指 I/O 接口中程序可访问的寄存器,有数据端口、命令端口和状态端口。通常用户程序不访问这些 I/O 端口,而由操作系统中的驱动程序访问。为了使指令能够访问到 I/O 端口,需要对它们进行编号,称为 I/O 端口编址(有时称为设备编址,实际上并不是对设备编址)。

有独立编址和统一编址两种方式。独立编址方式下,对 I/O 端口单独编号,使它们成为一个独立的 I/O 地址空间,此时,I/O 端口号可能和主存单元号相同,因此,从地址形式上无法区分指令访问的是 I/O 端口还是主存单元,需要通过不同的操作码来区分,因而需要



提供专门的I/O指令来控制对I/O端口的访问。统一编址方式下,I/O端口与主存地址空间统一编号,将主存地址空间分出一部分地址编号给I/O端口进行编号,因此,也称为存储器映射方式。因为主存单元和I/O端口在同一个地址空间,所以,主存单元号和I/O端口号肯定不会相同,它们分属两个不同的地址范围,因此,通过指令给出的地址范围就可以确定访问的是主存单元还是I/O端口,因而指令系统无须提供专门的I/O指令。

7. 常用I/O控制方式

目前,单处理器计算机系统中常用的I/O方式有程序直接控制、中断控制和DMA控制3种。

程序直接控制方式分无条件传送和条件传送方式。无条件传送方式利用程序定时传送数据,无须检测接口或设备的状态,适合各类巡回检测或过程控制;条件传送方式也称为程序查询方式,CPU通过查询外设接口中的"就绪(Ready)"、"忙(Busy)"和"完成(Done)"等状态来控制数据的传送,有定时查询和独占查询两种,独占查询方式下,CPU在整个数据交换过程中,一直为设备的I/O服务。

中断控制方式也是一种通过执行程序来进行数据交换的I/O方式。当外设准备好数据或准备好接收新数据或发生了特殊事件时,外设通过向CPU发中断请求来使CPU转到相应的中断服务程序去执行,在中断服务程序中完成数据交换或处理特殊事件。中断方式下,由硬件和软件共同完成整个中断过程。首先,由I/O接口向CPU发中断请求,CPU每执行完一条指令都去采样中断请求线,一旦发现有中断请求,并且处于"开中断(中断允许)"状态,CPU就进入"中断响应"周期,自动执行一条隐指令,完成关中断、保护断点、识别中断源3项任务,识别中断源的结果就是将中断服务程序的首地址送到PC中。"中断响应"周期结束,CPU就根据PC的值开始执行中断服务程序。对于单级中断系统,中断服务程序执行过程中一直不会开中断,直到中断返回前才执行"开中断"指令;而对于多级中断系统,中断处理过程可能被其他新中断打断,是否允许打断,通过在每个中断服务程序中设置中断屏蔽字来实现,中断服务程序中还要进行现场的保护和恢复。

DMA控制方式适合像磁盘一类的高速设备以成批方式和主存直接交换数据。首先要对DMA控制器进行初始化;然后由DMA控制器控制总线在主存和高速设备之间进行直接数据交换;最后,DMA控制器发出"DMA传送结束"信号给外设接口,由外设接口发中断请求给CPU,由CPU执行相应的中断服务程序来进行数据校验等,最终完成DMA传送处理。

9.3 基本术语解释

I/O带宽(I/O Bandwidth)

单位时间内系统输入/输出的数据量或所完成的I/O操作次数。即指在一定时间内所完成的工作量,也称为吞吐率(Throughput)。

响应时间(Response Time)

也称等待时间(Latency),是指从作业(或I/O请求)提交开始到作业(或I/O操作)完成所用的时间。

外设(Device)

外部辅助存储器和输入设备、输出设备统称为外设。也称为外围设备、外部设备或I/O

设备。

输入设备(Input Device)

输入设备的作用是将程序、原始数据、文字、图像、控制命令或现场采集的数据等信息输入到计算机。常见的输入设备有键盘、鼠标器、扫描仪等。

输出设备(Output Device)

输出设备把计算机的中间结果或最后结果、机内的各种数据符号及文字或各种控制信号等信息以某种形式输出到计算机外部。常用的输出设备有显示终端(CRT/LCD)、打印机、绘图仪等。

存储设备(Storage Device)

各种外部存储器称为存储设备,可以把信息从存储设备输入到主存,也可以把主存中的信息输出到存储设备上保存。如软盘、磁盘、光盘、磁带等。

终端(Terminal)

传统终端是指一种计算机外部设备,现在的终端概念已定位到一种由 CRT 显示器、控制器及键盘合为一体的设备,它与平常指的微型计算机的根本区别是没有自己的中央处理单元(CPU),当然也没有自己的内存,其主要功能是将键盘输入的请求数据发往主机(或打印机)并将主机运算的结果显示出来。对互联网而言,终端泛指一切可以接入网络的计算设备,如个人计算机、网络电视、可上网手机、PDA 等。

磁道(Track)

磁盘在高速旋转时,磁头对于磁盘表面做相对运动。相对运动时磁头在盘面上所经过的路径构成一个磁道。磁头在不同的位置,磁盘表面就形成不同半径的同心圆,因此,每个同心圆为一个磁道。每个磁道都有一个编号,最外面的是 0 磁道。

柱面(Cylinder)

在一个磁盘驱动器中的若干个盘片都连到同一个中心轴上,每个可读写的盘面上都有一个磁头,这些磁头被连在一起且同时按相同的轨迹移动。因此,在不同盘面上的磁头总是处在相同半径的磁道上,所有盘面上的这些磁道构成一个柱面。因此,柱面号和磁道号是一样的。磁头号和盘面号是一样的。

扇区(Sector)

每个磁道被划分为若干段,又叫扇区,每个扇区的存储容量均为 512 字节或 4096 字节。每个扇区都有一个编号,扇区是磁盘的最小编址单位。因此,到磁盘上寻找数据时,只需定位到相应的扇区。读写数据时可能会以多个扇区为单位进行传输。

道密度(Track Density)

沿磁盘半径方向单位长度上的磁道数称为道密度。道密度的单位是道/英寸(英文表示为 TPI,是 Tracks Per Inch 的缩写)或道/毫米(TPM)。道密度是相邻磁道间距的倒数。

位密度(Bit Density)

磁道上单位长度所记录的二进制信息位叫位密度或线密度,早期的磁盘,每个磁道所含信息量一样。这样的磁盘因为周长不同,其每个磁道的位密度也不同。一般磁盘的位密度是指最内圈上的位密度。单位是位/英寸或位/毫米。现在也有一些磁盘的磁道采用变长方式记录信息,外圈上的磁道可以比内圈上的磁道多记录一些信息。

磁盘平均存取时间(Average Access Time)

操作系统必须通过 3 个步骤对磁盘进行操作：寻道(Seek)、旋转(Rotation)、读写数据。

因为读写数据所用时间相对于前两个操作而言，可以忽略不计，所以，通常将前两个操作时间的平均值之和称为磁盘平均存取时间。即平均存取时间等于平均寻道时间加平均旋转时间。

平均寻道时间(Average Seek Time)

移动磁头，使磁头定位到要读写的磁道上的操作称为寻道(Seek)。平均寻道时间取决于相邻两道之间的寻道时间，称为道间移动时间，工业上也把它称为最小寻道时间。平均寻道时间有很多种测量方法，最简单的方法就是把最长寻道时间除以2。最长寻道时间就是从最内(外)移过所有磁道到最外(内)磁道的时间。

平均旋转时间(Average Rotational Latency/Delay)

磁头定位在要读写的磁道后，磁盘开始旋转直到磁头正好落在要读写的数据上方。通常把完成这个过程的时间称为旋转(等待)时间。平均旋转时间是磁盘转一圈的时间的一半。

传输时间(Transfer Time)

当磁头正好落在要读写数据的起点后，就开始读/写数据，磁盘读/写一块数据(一个扇区)的时间为传输时间。它是扇区大小、旋转速度、磁道上位密度的函数。

磁盘数据传输率(Transfer Rate)

分为外部数据传输率和内部数据传输率两种。外部数据传输率指主机接口从(向)硬盘缓存读出(写入)数据的速度；内部数据传输率指读写磁头定位在要读写的数据块的始端后，单位时间内连续从磁道中读出或写入的二进制数据的位数。

磁盘控制器(Disk Controller)

指用来控制磁盘进行数据读写，并控制数据在磁盘和主存间进行传输的部件。因此，一次磁盘读写操作除了寻道、旋转、传输3个操作时间以外，还要加上磁盘控制器所用的时间。

廉价磁盘冗余阵列(Redundant Arrays of Inexpensive Disk,RAID)

将多个独立操作的磁盘按某种方式组织成磁盘阵列，以增加容量，利用类似于主存中的多体交叉技术，将数据存储在多个盘上，通过使这些盘并行工作来提高数据传输速度，并用冗余磁盘技术提高系统可靠性。

I/O 接口(I/O Interface,I/O Module)

I/O 接口(I/O 模块)是主机和外设之间传送信息的“桥梁”，介于主机和外设之间。主机控制外设的命令信息、传送给外设的数据或从外设取来的数据、外设送给主机的状态信息等都要先存放到 I/O 接口。对每种具体的设备来说，就是介于底板总线和通信总线(电缆式总线)之间的扩展卡或插件板，例如，网卡、显示卡等。也有很多设备的接口电路集成在主板上，如声卡、Modem 卡、打印机控制卡、磁盘控制器、键盘/鼠标控制电路等已经基本上集成在主板上，只留一些电缆插座，以连接相应的外部设备。

I/O 控制器(I/O Controller)

I/O 接口中的控制电路，不包含 I/O 接口中的连接器插座。

I/O 端口(I/O Port)

I/O 接口中一些可被程序访问的寄存器，用来存放控制(命令)、数据和状态信息，这些寄存器被称为 I/O 端口。

命令(控制)端口(Command/Control Port)

在 I/O 接口中,用来存放 CPU 送来的控制信息的寄存器称为命令端口或控制端口,只可以对其进行写操作。

数据端口(Data Port)

在 I/O 接口中,用于存放接收和发送数据的寄存器称为数据端口,可以对其进行读或写操作。

状态端口(Status Port)

在 I/O 接口中,用来记录设备或 I/O 接口的状态的寄存器称为状态端口,通常只可以对其进行读操作。有些接口电路,将命令端口和状态端口合二为一,作为命令端口时,从 CPU 写入控制信息,作为状态端口时,存入外设或接口的状态信息,供 CPU 读取。

I/O 地址空间(I/O Address Space)

所有 I/O 端口号组成的地址空间,也简称为 I/O 空间。

独立编址(Special Address Space)

将 I/O 端口和主存单元分别编号,不占用主存单元的地址空间,因而主存单元和 I/O 端口可能会有相同的编号,但地址位数大多不同,主存单元多,地址空间大,因而地址位数多;I/O 端口少,地址空间小,因而地址位数少。因为可能有相同的编号,指令中无法靠地址来区分要访问的是主存单元还是 I/O 端口,所以,需要有和访存指令不同的操作码,因此,需要设计专门的 I/O 指令。

统一编址(United Address Space)

I/O 端口和主存单元统一编址,所以也称为存储器映射 I/O(Memory-mapped I/O)方式。一个地址空间分成了两部分,各在不同的地址范围内,但地址的位数是相同的,可根据地址范围的不同来区分访问的是主存单元还是 I/O 端口,所以无须专门地输入输出指令。

并行接口(Parallel Interface)

并行接口在设备和接口之间同时传送一个字节或一个字的所有位。

串行接口(Serial Interface)

串行接口在设备和接口之间按位来传送数据。

可编程接口(Programmable Interface)

能用程序来改变或选择接口的功能和操作方式。

不可编程接口(Non-programmable Interface)

不能用程序来改变其功能和操作方式,但可通过硬连线路来实现不同的功能。

I/O 指令(I/O Instruction)

指用来访问 I/O 端口的指令。指令中的操作码必须指出是读还是写,地址码必须说明信息在哪个端口和哪个通用寄存器之间进行传送,所以,地址码中要给出端口号和通用寄存器编号。如 80x86 中的 IN 指令和 OUT 指令等。

在具有通道的大型机中,CPU 通过通道控制外设,此时 I/O 指令是指控制通道和外设控制器的指令。如 START I/O、TEST I/O、HALT I/O 指令等。

程序查询 I/O 方式(I/O Polling)

CPU 通过执行查询程序来完成对外设的控制,并实现和外设之间的数据传送。在查询程序中,CPU 首先通过读取状态端口中的状态信息,了解接口是否已“就绪”(或“完成”),是

的话就通过数据端口进行新的数据传送，并查询外设是否空闲，在外设空闲的情况下，通过发送控制信息到命令端口，然后由接口发"启动"命令送外设；如果接口没有就绪，或外设不空闲，则 CPU 继续查询，以等待接口就绪或外设空闲。所有信息(包括：控制、数据和状态信息)的交换由查询程序中的 I/O 指令完成。

就绪状态(Ready)

对于输入设备而言，就绪状态意味着 I/O 接口中的数据缓冲器的数据已有效，CPU 可以从 I/O 接口取数据；对于输出设备而言，则说明 I/O 接口中的数据缓冲器已空，CPU 可以将数据送到 I/O 接口中。

程序中断 I/O 方式(I/O Interrupt)

程序中断 I/O 方式下，CPU 启动外设后，就转到另外一个程序执行，此时，外设和 CPU 并行工作。一旦外设完成任务，便发中断请求给 CPU，告知 CPU 刚才所请求的 I/O 任务已经完成。此时，CPU 暂停正在执行的程序，转到一个中断服务程序进行中断处理，在中断处理过程中，进行外设 I/O 下一步的准备工作(如：传送下一个要打印的数据；取走键盘数据或采样数据，为下次输入腾空数据缓冲寄存器；等等)，最后启动外设，并回到原程序继续执行。此时，CPU 和外设又能并行工作。

可屏蔽中断(Maskable Interrupt)

如果某个中断事件不是很紧急，可以被延缓响应，那么在处理其他中断时，就可以屏蔽这个中断，让正在处理的中断执行完后，再来响应这个中断。这种称为可屏蔽中断。一般外设中断源引起的中断都是可屏蔽中断。

不可屏蔽中断(Non-maskable Interrupt，NMI)

不可屏蔽中断是指重要或紧急的硬件故障事件，如电源掉电、存储器线路出错等。这类中断发生时，必须立即响应，不能把这类中断屏蔽掉。

中断请求寄存器(Interrupt Request Register)

中断控制器中专门用来存放每个设备对应的中断请求信号的寄存器。

中断屏蔽寄存器(Interrupt Mask Register)

中断控制器中专门用来存放每个中断源对应的中断屏蔽字的寄存器。

中断响应优先级(Interrupt Response Priority)

中断响应优先级是指当多个中断请求发生时，优先响应哪个中断请求。它是由硬件排队线路或中断查询程序的查询顺序决定的，不可动态改变。

中断处理优先级(Interrupt Process Priority)

中断处理优先级反映的是正在处理的中断是否比新发生的中断的处理优先级低(即：屏蔽位为"0"，对新中断开放)，如果是的话，就中止正在处理的中断，转到新的中断服务程序去执行，处理完后回到原被中止的中断服务程序继续执行。中断处理优先级可以由中断屏蔽字来动态改变。

DMA(直接存储器存取)I/O 方式(Direct Memory Access)

DMA 是 Direct Memory Access 的缩写。每次需要进行外设数据读写时，首先 CPU 把要传送的数据个数、数据块在内存的首址、数据传送的方向(是读操作还是写操作)、设备的地址等参数送给 DMA 控制器，最后发送一个命令给 DMA 接口，以启动外设进行数据传送。在这些工作完成后，CPU 就继续进行其他工作。此时，CPU 和外设并行工作，而外设

和主存交换数据的工作就交给 DMA 控制器控制完成。DMA 控制器在需要时申请总线控制权，占用总线完成外设和主存间的数据传送。传送结束后，向 CPU 发送“DMA 结束”中断请求，让 CPU 进行数据校验等后处理工作。

DMA 方式适合像磁盘一类的高速设备，以成批的方式和主存直接交换数据。

周期挪用(Cycle Stealing)

周期挪用法的基本思想是，当外设准备好一个数据时，DMA 控制器就向 CPU 申请一次总线控制权，CPU 在一次总线操作结束时一旦发现有 DMA 请求，就立即释放总线，让出一个周期给 DMA 控制器，由 DMA 控制器控制总线在主存和外设之间传送一个数据，传送结束后立即释放总线，下次外设准备好数据时，又重复上述过程，直到所有数据传送完毕。在这种情况下，CPU 的工作几乎不受影响，只是在万一出现访存冲突(即 CPU 和 DMA 控制器同时要求访问同一个主存)时，CPU 挪出一个周期给 DMA，由 DMA 访问主存，而 CPU 延迟访问主存。这里 CPU 挪用的是主存存取周期。

DMA 控制器(DMA Controller)

把 DMA 接口中控制数据传送的硬件逻辑称为 DMA 控制器。它能像 CPU 一样控制总线，完成 I/O 设备和主存间的数据传送。

通道(Channel，CH)

通道是用来负责管理外设以及实现主存和外设之间数据交换的一种专门的控制部件。通道能够执行专门的通道指令，若干条通道指令组成一个通道程序，通过执行通道程序进行 I/O 操作。CPU 通过执行专门的 I/O 指令来启动、停止通道，或测试或改变通道工作状态，而不直接参与对外设的管理和控制。

通道指令(Channel Command)

又称为通道控制字(Channel Control Word，CCW)，是为通道专门设计的用于 I/O 的指令。一个通道控制字一般包括操作命令码、数据在内存的首址、传送数据个数和控制标志字段等。

输入/输出处理机(I/O Processor)

输入/输出处理机方式是通道方式的进一步发展，有两种输入/输出处理机系统结构。一种是通道结构的输入/输出处理机，通常称为 I/O 处理机(IOP)。另一种输入/输出处理机系统结构是外围处理器(PPU)方式。在大型计算机系统中，有时选用通用计算机担任 PPU，它基本上独立于主 CPU 而工作，也有自己的指令系统，可进行算术/逻辑运算、主存读写和与外设交换信息等。

9.4 常见问题解答

1. I/O 设备和 I/O 接口两部分结合起来就是输入输出系统吗？

答：不是。I/O 设备和 I/O 接口只是 I/O 硬件部分，输入输出系统应该包括 I/O 硬件和 I/O 软件两个部分。不同硬件结构的 I/O 系统，所采用的 I/O 软件技术差别很大。但不管是哪种，CPU 通过直接执行 I/O 指令或操作系统管理程序，总是或多或少地参与主机和外设交换信息的任务，也就是说输入/输出任务总要有 I/O 软件的参与。

2. I/O 系统的性能如何评价？

答：一般用响应时间和吞吐率两个指标来衡量。不同的 I/O 系统对于响应时间和吞吐

率的要求不同。例如，对于事务处理系统(如：订票、存/取款等系统)，由于同时会有大量的事务要求处理，且每个事务对磁盘的访问量很少，所以这种系统主要考虑每秒钟磁盘的存取次数能否达到很大，使得同时有很多事务在很短的时间内得到快速响应。也就是说，对响应时间的要求更高，而不大在乎吞吐率。但是，像多媒体点播系统，就希望系统的吞吐量很大，要求单位时间内能有大量数据读出，以满足播放要求。

3. 数据传输率中的K、M、G等单位的含义和主存容量中的含义一样吗？在磁盘容量中的含义呢？

答：不一样。在主存容量中 $1K=2^{10}$、$1M=2^{20}$、$1G=2^{30}$。但是，在数据传输率中，因为数据传输速度和时钟频率有关，时钟频率通常以kHz、MHz、GHz来表示，所以，传输速率一般用kB/s、MB/s、GB/s来表示，这里的 $1k=10^3$、$1M=10^6$、$1G=10^9$。计算中可能会混合在一起，因为，数据块的大小还是用 $1K=2^{10}$、$1M=2^{20}$、$1G=2^{30}$，而传输速率又和数据块大小有关，这样使得计算变得很复杂。

磁盘容量以兆字节(MB)或千兆字节(GB)等为单位，$1GB=1024MB=2^{30}B$。但硬盘厂商在标称硬盘容量时通常取 $1G=1000M=10^9$，因此在BIOS或在硬盘格式化时标出的容量会比厂家标称值小。

4. 什么是I/O设备的数据速率？各种外设的数据速率相差很大吗？

答：I/O设备的数据速率是指在I/O设备和主机之间传输数据时的峰值速率。各种I/O设备的数据速率相差很大。有键盘、鼠标这种极慢速的设备，大约每秒钟10个字节，也有磁盘和网络设备等这些每秒能达到几十兆字节的快速设备。不同传输速率的外设所用的外设接口不同，在主板上的连接方式不同，所用的I/O方式也不同。

5. 磁盘上信息是如何组织的？磁盘的最小编址单位是什么？

答：磁盘表面被分为许多同心圆，每个同心圆被称为一个磁道。信息存储在磁道上。每个磁道被划分为若干段，称为扇区。每个扇区存放一个记录块，每个记录块有相应的地址标识字段、数据字段和校验字段等。到磁盘上寻找数据时，只要定位数据所在的磁头、磁道和扇区。所以，扇区是磁盘的最小编址单位。

6. 一个磁盘扇区的大小总是512字节吗？

答：不是。近三十年来，一个硬盘扇区的大小一直都是512字节。但是，最近几年，硬盘制造商正在从传统的512字节扇区迁移到更大、更高效的4096字节扇区，通常称为4K扇区。

7. 盘面号和磁头号是一回事吗？

答：是的。硬盘是一个盘组，由多个盘片组成，每个盘片有两个面。每个盘面上有一个磁头，用于对该盘面上的信息进行读写。所以，磁头号就是盘面号。

8. 柱面号和磁道号是一回事吗？

答：是的。硬盘是一个盘组，由多个盘面组成，所有盘面上相同编号的磁道构成了一个圆柱面(但物理上这个圆柱面是不存在的)，有多少磁道就形成多少圆柱面。所以，磁道号就是圆柱面号。

9. 当一个磁道存满后，信息是在同一个盘面的下一个磁道存放，还是在同一个柱面的下一个盘面存放？

答：当一个磁道存满后，如果信息是在同一个盘面的下一个磁道存放，则需要移动磁

头，因为移动磁头是机械运动，所以花费时间较长，且有机械磨损；如果信息在同一个柱面的下一个盘面存放，则不需移动磁头，即磁道号不变，只要给出一个相邻盘面号，通过译码电路选取该盘面的磁头就可以读写，几乎没有延迟，也没有机械运动。

10. I/O 接口就是 I/O 端口吗？

答：不是。I/O 接口和 I/O 端口是两个不同的概念，但相互之间有关联。I/O 接口是主机和外设之间传送信息的“桥梁”，介于主机和外设之间。主机控制外设的命令信息、传送给外设的数据或从外设取来的数据、外设送给主机的状态信息等都要先存放到 I/O 接口中，所以，接口中有一些寄存器，用于存放这些命令、数据和状态信息。把 I/O 接口中的这些寄存器称为 I/O 端口。

11. I/O 端口是如何编址的？

答：一般有两种编址方式：独立编址和统一编址。这里的“统一”和“独立”不是指各个不同接口之间的“统一”和“独立”关系，而是指所有 I/O 端口号组成的地址空间（I/O 地址空间）和所有主存单元号组成的地址空间（主存地址空间）之间的关系。

12. CPU 是如何访问 I/O 端口的？

答：在 I/O 指令中给出要访问的端口号，当 CPU 执行 I/O 指令时，根据指令的操作码或地址范围得知要访问 I/O 地址空间，因而在总线的地址线上送出端口号，在总线的控制线上送出 I/O 读或 I/O 写命令，被访问端口所在的接口电路对地址译码后选中相应的端口，并从控制线上取得读/写命令，由接口中的读/写控制电路对被访问端口进行读或写操作。

13. 一个 I/O 接口只能有一个地址吗？

答：不是。每个 I/O 端口对应唯一的一个地址，但一个 I/O 接口中可能有多个程序可访问的寄存器，也就是有多个 I/O 端口，所以应该有多个地址。

14. 程序查询方式下，外设的数据是直接和 CPU 交换的吗？

答：是的。程序查询方式下，整个输入输出过程是通过 CPU 执行查询程序完成的，所有信息（命令、数据、状态）的交换具体由查询程序中的 I/O 指令进行控制，因而外设的数据是直接和 CPU 交换的。

外设的数据和状态信息通过 I/O 接口中设备侧的电缆线（通信总线）送到 I/O 接口中，连同接口本身的状态信息一起记录到相应的数据或状态端口中，CPU 再通过执行输入指令（如 80x86 中的 IN 指令）从 I/O 端口中将状态或数据取到 CPU 的寄存器中。CPU 送到外设的数据和命令字，通过执行输出指令（如 80x86 中的 OUT 指令）从 CPU 中的寄存器送到相应的 I/O 端口中。

15. 中断方式下，外设的数据是直接和 CPU 交换的吗？

答：是的。中断方式下，当外设完成任务（如打印完一个字符、键盘有按键等），或外设发生了特殊事件（如打印机缺纸、过程控制中温度太高、采样定时到等），外设通过向 CPU 发中断请求来中止 CPU 正在执行的程序，转到相应的中断服务程序去执行，处理完后，回到原被中止的程序继续执行。通常在 CPU 执行中断服务程序过程中完成数据的交换，如从键盘缓冲取数据，向打印机缓冲发送打印字符，取采样数据等。这些都是通过 CPU 执行 I/O 指令来完成的，因而，对于采用中断方式的 I/O 过程，外设的数据是直接和 CPU 交换的。

16. DMA方式下，外设的数据是直接和CPU交换的吗？

答：不是。DMA方式适合像磁盘一类的高速设备，以成批方式交换几百到几千字节数据，CPU不可能放得下那么多数据。所以，DMA方式下，设备直接和主存进行数据交换，由专门的硬件(DMA控制器)控制在主存和外设之间进行数据传送。

17. 中断方式下，外设任何时候都可以申请中断并马上得到响应吗？

答：不是。中断方式下，外设何时发出中断请求是由外设接口中的中断逻辑决定的，不受CPU的限制，但何时响应中断与CPU执行指令过程有关。CPU总是在一条指令执行完、取下条指令之前去查询有无中断请求。如果此时是开中断状态，并有未被屏蔽的中断请求发生，则CPU自动执行一条隐指令，进行中断响应，完成关中断、保护断点、取中断向量3个操作。因此，不是任何时候都马上响应中断，中断响应的条件有3个：(1)CPU处于开中断状态(中断允许触发器EINT置“1”状态)；(2)至少有一个未被屏蔽的中断请求发生；(3)一条指令执行结束。

18. 为什么在介绍CPU设计时要讲中断的概念，在介绍I/O系统时又讲中断的概念？

答：在CPU执行程序过程中，有两种情况会打断程序的执行，一种情况是CPU正在执行的指令出现了“异常”或设置了“陷阱”；另一种情况是指令执行正常，但外部设备出现了特殊事件，要求CPU处理。一般把前者称为“异常”，后者称为“中断”(也有很多系统或教科书不分“异常”和“中断”，全部称为“中断”)。

在进行CPU设计时，必须考虑在数据通路中如何实现异常和中断处理，包括：如何设置“开/关中断”状态，如何判断是否发生了异常和中断、如何识别是哪类异常和中断、怎样保存断点、如何切换到中断服务程序等。因此，在第6章介绍CPU设计时出现了异常和中断的概念。

同时，“中断”作为一种I/O方式，在许多采用中断方式的外设接口电路中，必须要有相应的中断处理逻辑，因此，在本章涉及I/O系统设计时，也谈到了很多有关中断的概念。

19. 为什么在响应中断的时候保存断点，而在处理中断的时候保存现场？

答：断点是中断返回到被中断程序继续执行处指令的地址(即：响应中断时PC的值)，必须在中断响应时先保存到栈中，否则，当取来中断服务程序的首地址送PC后，原来作为断点的PC的值就被破坏了。而现场是被中断的原程序在断点处各个寄存器的值，只要在这些寄存器再被使用前保存到栈中就行了，因为在实际处理中断事件过程中可能要用到这些寄存器，所以在具体中断处理之前的准备阶段来保存现场(寄存器压栈)，而在具体处理后的结束阶段再恢复现场(寄存器出栈)。这样就能保证被中断程序的现场不被中断服务程序破坏。

20. 单重中断和多重中断的区别是什么？

答：单重中断情况下，在中断处理整个过程中，不允许响应新的中断请求，其做法是在中断响应开始时关中断(使中断允许触发器置“0”)，而直到中断处理结束后才开中断，然后返回到原断点处继续执行。

多重中断系统中，如果在进行某个具体的中断处理过程中，又发生了新的中断请求，则可以中止正在进行的中断处理，转到新的中断服务程序执行。因此，在中断处理过程中，应该开中断，允许响应新的中断请求。其做法是在实际处理中断事件前就开中断，而不是像单重中断那样在处理后才开中断。这样保证在实际中断处理过程中可以响应新的中断请求。

21. 向量中断、中断向量、向量地址3个概念是什么关系?

答:中断向量:每个中断源都有对应的处理程序,称这个处理程序为中断服务程序,其入口地址称为中断向量。所有中断的中断服务程序入口地址构成一个表,称为中断向量表;有的机器也把中断服务程序入口的跳转指令构成一张表,称为中断向量跳转表。

向量地址:中断向量表或中断向量跳转表中每个表项所在的主存地址或表项的索引值,称为向量地址或中断类型号。

向量中断:是指一种识别中断源的技术或方式。识别中断源的目的就是要找到中断源对应的中断服务程序的入口地址,即获得向量地址。采用向量中断进行中断源识别的做法如下:通过某种硬件排队线路(如:菊花链、并行判优等),对所有未被屏蔽的中断请求进行排队,选出优先级最高的中断源,然后对其编码,得到该中断源的编号(可以转换为向量地址,有些书中就称其为向量地址)。通过总线将其取到CPU中,并转换成向量地址,从而取出中断服务程序入口地址,或跳转到中断服务程序。还有一种是用程序(中断查询程序)进行中断源识别的软件方法。

22. 禁止中断和屏蔽中断是同一个概念吗?

答:它们是两个完全不相关的概念。

禁止中断就是关中断,即将中断允许触发器置为“0”,此时,任何中断请求都得不到响应。屏蔽中断是多重中断系统中的一个概念,是指某个中断正在被处理时,如果有其他新的中断请求发生,那么,通过设置中断屏蔽位,可确定是否允许响应新发生的中断请求。它反映了正在处理的中断与其他各中断之间的处理优先级顺序,所以每个中断都有一个中断屏蔽字,其中的每一位对应一个中断的屏蔽位。响应某个中断后,就会把该中断的中断屏蔽字送到中断屏蔽字寄存器中,在中断排队前,其中的每一位和中断请求寄存器中的对应位进行“与”操作,因而,只有未被屏蔽的中断源进入排队线路,从而有可能得到响应。

23. 中断响应优先级和中断处理优先级一样吗?

答:不一样,这是两个不同的概念。中断响应优先级是由硬件排队线路或中断查询程序的查询顺序决定的,不可动态改变;而中断处理优先级可以由中断屏蔽字来改变,反映的是正在处理的中断是否比新发生的中断的处理优先级低,如果是,就中止正在处理的中断,转到新中断去处理,处理完后回到原被中止的中断继续处理。

24. DMA方式下,在主存和外设之间有一条物理通路直接相连吗?

答:没有。通常所说的DMA方式下数据在主存和外设之间直接进行传送,其含义并不是说在主存和外设之间建立一条物理上的直接通路,而是在主存和外设之间通过外设接口、系统总线以及总线桥接部件等连接,建立起一个信息可以互相通达的通路。“直接通路”是逻辑上的含义,物理上磁盘和主存不是直接相连的。

25. DMA方式下,CPU一点开销都没有吗?

答:不是。DMA方式下的数据交换过程分以下3个步骤:

(1) DMA控制器的初始化。将所要传送的数据个数、内存地址、传送方向等送到DMA控制器。这个过程由CPU执行指令来完成。初始化结束后,CPU发送启动磁盘定位和DMA传送的命令,这也是通过CPU执行输出指令来完成的。

(2) DMA传送。这个过程整个都是由DMA控制器来完成的,主要由DMA控制器控制系统总线,完成数据在主存和外设之间的数据传送。

(3) DMA 传送结束处理。DMA 传送结束后，向 CPU 发出“DMA 结束”中断请求，由 CPU 执行相应的中断服务程序进行数据校验等后处理工作。

综上所述，DMA 方式下，CPU 要进行初始化和后处理两部分工作，因此，不是一点开销都没有，只是相对于程序查询方式和中断方式来说，CPU 介入要少得多，因为 CPU 不需要介入主要的数据传送过程。

26. CPU 对 DMA 请求和中断请求的响应时间是否一样?

答：不一样。DMA 方式下，向 CPU 请求的是总线控制权，要求 CPU 让出总线控制权给 DMA 控制器，由 DMA 控制器来控制总线完成主存和外设之间的数据交换，因此，CPU 只要用完总线后就可以响应请求，释放总线，让出总线控制权。CPU 总是在一次总线事务完成后响应，因此，DMA 响应时间应该少于一个总线周期；而中断方式下请求的是 CPU 时间，要求 CPU 中止正在执行的程序，转到中断服务程序去执行，通过执行中断服务程序，对中断事件进行相应的处理。CPU 总是要等到一条指令执行结束后，才去查询有无中断请求，所以响应时间少于一个指令周期的时间。

27. 挪用周期方式下，DMA 控制器窃取的是什么周期?

答：周期挪用法的基本思想是，当外设准备好一个数据时，DMA 控制器就向 CPU 申请一次总线控制权，CPU 在一个总线事务结束时一旦发现有 DMA 请求，就立即释放总线，让出一个周期给 DMA 控制器，由 DMA 控制器控制总线在主存和外设之间传送一个数据，传送结束后立即释放总线，下次外设准备好数据时，又重复上述过程，直到所有数据传送完毕。这种情况下，CPU 的工作几乎不受影响，只是在万一出现访存冲突时，CPU 挪出一个周期给 DMA，由 DMA 访问主存，而 CPU 延迟访问主存。这里 CPU 挪出的是主存的存储周期。

9.5 单项选择题

1. 假定有一个事务处理系统 A，其处理器的速度为每秒钟执行 5 千万条指令，每个事务需要 5 次 I/O 操作，每次 I/O 操作需要 10000 条指令。如果系统 A 每秒钟最多完成 1000 次 I/O 操作，则它每秒钟处理的事务数最多能达到(　　)个。(忽略延迟并假定事务可以不受限制地并行处理)

   A. 200　　B. 1000　　C. 2000　　D. 10000

2. 以下各类外设中，属于成块传送设备的是(　　)。

   A. 键盘　　B. 鼠标器　　C. 针式打印机　　D. U 盘

3. 以下各类外设中，属于存储设备的是(　　)。

   A. 键盘　　B. 鼠标器　　C. 显示器　　D. 磁带机

4. 以下各类外设中，属于字符型设备的是(　　)。

   A. Modem　　B. 硬盘存储器　　C. 光驱　　D. 软驱

5. 以下是有关非编码键盘和鼠标器的描述：

   ① 键盘和鼠标都是字符型输入设备
   ② 键盘和鼠标都以串行方式和主机通信
   ③ 键盘和鼠标都采用中断方式进行数据传送
   ④ 键盘和鼠标向主机传送的都是位置信息

以上描述中,正确的是(　　)。

A. ①、②　　B. ①、②、③　　C. ①、③、④　　D. 全部

6. 在采用中断方式进行打印控制时,在打印控制接口电路和打印部件之间交换的信息不包括以下的(　　)。

A. 打印字符点阵信息　　B. 打印控制信息

C. 打印机状态信息　　D. 中断请求信号

7. 以下是有关激光打印机中打印控制器功能的叙述,其中错误的是(　　)。

A. 接收主机送来的打印字符点阵信息

B. 缓存主机送来的各种打印描述信息

C. 对打印语言中描述的各种命令进行解释

D. 对打印内容按页面设置参数进行光栅化

8. 假定一台计算机的显示存储器用 DRAM 芯片实现,若要求显示分辨率为 1600×1200,颜色深度为 24 位,帧频为 85Hz,显存总带宽的 50%用来刷新屏幕,则需要的显存总带宽至少约为(　　)。

A. 245Mbps　　B. 979Mbps　　C. 1958Mbps　　D. 7834Mbps

9. 随着 3D 游戏软件的大量出现,许多机器的显卡中配置了具有 3D 功能的 GPU,以下有关这类显卡的显示存储器(VRAM,显存)的叙述中,错误的是(　　)。

A. 显存大多由 DRAM 芯片组成

B. CPU 和 GPU 都可以访问显存

C. 显存和主存之间可采用 DMA 方式传送数据

D. 显存容量大致等于最高分辨率与像素深度的乘积

10. 以下有关磁盘驱动器的叙述中,错误的是(　　)。

A. 送到磁盘驱动器的盘地址由磁头号、盘面号和扇区号组成

B. 能控制磁头移动到指定磁道,并发回“寻道结束”信号

C. 能控制磁盘片转过指定的扇区,并发回“扇区符合”信号

D. 能对指定盘面的指定扇区进行数据的读或写操作

11. 假定一个磁盘存储器有 4 个盘片,用于记录信息的柱面数为 2000,每个磁道上有 3000 个扇区,每个扇区 512B,则该磁盘存储器的容量约为(　　)。

A. 12MB　　B. 24MB　　C. 12GB　　D. 24GB

12. 假定一个磁盘的转速为 7200RPM,磁盘的平均寻道时间为 20ms,平均数据传输率为 1MB/s,不考虑排队等待时间。那么读一个 512 字节的扇区的平均时间大约为(　　)。

A. 14.7ms　　B. 18.8ms　　C. 24.7ms　　D. 28.8ms

13. 假定计算机系统中连接主存和磁盘的总线带宽是 68.8MB/s,磁盘的最大数据传输率是 5MB/s。如果允许磁盘输入/输出占用 100%的总线和主存带宽,那么总线上可同时接入的磁盘个数最多是(　　)。

A. 13　　B. 14　　C. 15　　D. 16

14. 以下有关 RAID 技术的叙述中,错误的是(　　)。

A. RAID 技术可实现海量后备存储系统

B. RAID 技术可提高存储系统的可靠性

C. RAID中的校验信息都存放在一个磁盘上

D. RAID通过多个盘并行访问来提高速度

15. 以下关于光盘存储器的叙述中,错误的是(　　)。

A. 光盘的数据记录在一条连续的螺旋状光道上

B. 光盘读/写信息时可采用恒定线速度方式进行旋转

C. 光盘上凹坑内的信息为0,凹坑外的信息为1

D. 光盘上存储信息的地址用时间单位来表示

16. 主机和外设之间的正确连接通路是(　　)。

A. CPU和主存—I/O总线—I/O接口(外设控制器)—通信总线(电缆)—外设

B. CPU和主存—I/O接口(外设控制器)—I/O总线—通信总线(电缆)—外设

C. CPU和主存—I/O总线—通信总线(电缆)—I/O接口(外设控制器)—外设

D. CPU和主存—I/O接口(外设控制器)—通信总线(电缆)—I/O总线—外设

17. 以下有关I/O接口功能和结构的叙述中,错误的是(　　)。

A. I/O接口就是像显卡或网卡之类的一种外设控制逻辑

B. CPU可以向I/O接口传送用来对设备进行控制的命令

C. CPU可以从I/O接口取状态信息,以了解接口和外设的状态

D. I/O接口中主机侧数据宽度与设备侧数据宽度总是一样

18. 以下有关I/O端口的叙述中,错误的是(　　)。

A. I/O接口中程序可访问的寄存器被称为I/O端口

B. I/O接口中有命令端口、状态端口和数据端口

C. I/O端口可以和主存统一编号,也可以单独编号

D. I/O接口中命令端口和状态端口不能共用同一个

19. 以下给出的部件中,不包含在外设控制接口电路中的是(　　)。

A. 标志寄存器　　　　B. 数据缓存器

C. 命令(控制)寄存器　　　　D. 状态寄存器

20. 以下有关统一编址方式的描述中,错误的是(　　)。

A. I/O端口地址和主存地址一定不重号

B. CPU通过执行访存指令来访问I/O端口

C. 必须根据指令类型区分访问主存还是访问I/O端口

D. 可利用主存的存储保护措施对I/O端口进行存储保护

21. 点对点I/O接口只能连接一个外设,而总线式多点I/O接口可连接多个外设。以下给出的几类I/O接口中,不是总线式多点I/O接口的是(　　)。

A. USB　　B. RS-232　　C. SCSI　　D. IEEE 1394(火线)

22. 以下给出的几类I/O接口中,可以采用并行传输方式的是(　　)。

A. USB　　B. RS-232　　C. SCSI　　D. IEEE 1394(火线)

23. 某终端通过串行通信接口与主机相连,采用起止式异步通信方式,若传输速率为9600波特,采用两相调制技术。通信协议为7位数据、1位校验位、1位停止位。传送一个字符所需时间约为(　　)。

A. 0.94ms　　B. 1.88ms　　C. 1.04ms　　D. 2.08ms

24. 以下 I/O 控制方式中，主要由硬件而不是软件实现数据传送的方式是(　　)。
    A. 程序查询方式　　B. 程序中断方式
    C. DMA 方式　　D. 无条件程序控制方式
25. 以下是有关程序直接控制(查询)I/O 方式的叙述：
    ① 无条件传送接口中不记录状态，无须状态查询，可直接定时访问
    ② 条件传送接口中有“就绪”、“完成”等状态，可定时查询或独占查询
    ③ 通过 CPU 执行相应的无条件传送程序或查询程序来完成数据传送
    ④ 适合巡回检测采样系统或过程控制系统，以及非随机启动的字符型设备
    以上叙述中，正确的有(　　)。
    A. ①、②　　B. ①、②、③　　C. ①、③、④　　D. 全部
26. 下列选项中，能引起外部中断请求的事件是(　　)。
    A. 鼠标输入　　B. 除数为 0
    C. 浮点运算下溢　　D. 访存缺页
27. 下列选项中，不属于外部中断的事件是(　　)。
    A. 采样定时到　　B. 无效操作码
    C. 打印机缺纸　　D. 键盘缓冲满
28. 下列选项中，能引起外部中断请求的事件是(　　)。
    A. 一条指令执行结束　　B. 一次总线传输结束
    C. 一次中断处理结束　　D. 一次 DMA 操作结束
29. 以下(　　)情况出现时，会引起 CPU 自动查询有无中断请求，进而可能进入中断响应周期。
    A. 一条指令执行结束　　B. 一次 I/O 操作结束
    C. 一次中断处理结束　　D. 一次 DMA 操作结束
30. 以下有关 CPU 响应外部中断请求的叙述中，错误的是(　　)。
    A. 每条指令结束后，CPU 都会转到“中断响应”周期进行中断响应处理
    B. 在“中断响应”周期，CPU 先将中断允许触发器清“0”，以使 CPU 关中断
    C. 在“中断响应”周期，CPU 把后继指令地址作为返回地址保存在固定地方
    D. 在“中断响应”周期，CPU 把取得的中断服务程序的入口地址送 PC
31. 单级中断系统中，中断服务程序内的执行顺序是(　　)。
    Ⅰ. 保护现场　　Ⅱ. 开中断　　Ⅲ. 关中断　　Ⅳ. 保存断点
    Ⅴ. 中断事件处理　　Ⅵ. 恢复现场　　Ⅶ. 中断返回
    A. Ⅰ→Ⅴ→Ⅵ→Ⅱ→Ⅶ　　B. Ⅲ→Ⅰ→Ⅴ→Ⅶ
    C. Ⅲ→Ⅳ→Ⅴ→Ⅵ→Ⅶ　　D. Ⅳ→Ⅰ→Ⅴ→Ⅵ→Ⅶ
32. 中断向量地址是指(　　)。
    A. 子程序入口地址　　B. 中断服务程序入口地址
    C. 中断服务程序入口地址的地址　　D. 中断查询程序的入口地址
33. 以下操作中，由硬件完成的是(　　)。
    A. 保护断点　　B. 保护现场
    C. 设置中断屏蔽字　　D. 从 I/O 接口取数

34. 设置中断屏蔽字可以动态地改变(　　)优先级。
A. 中断查询　　B. 中断响应　　C. 中断处理　　D. 中断返回

35. 开中断和关中断两种操作都用于对(　　)进行设置。
A. 中断允许触发器　　B. 中断屏蔽寄存器
C. 中断请求寄存器　　D. 中断向量寄存器

36. 以下有关中断 I/O 方式的叙述中,错误的是(　　)。
A. 中断请求的是 CPU 时间,要求 CPU 执行程序来处理发生的相关事件
B. CPU 对外部中断的响应不可能发生在一条指令的执行过程中
C. 中断 I/O 方式下,外设接口中的数据和 CPU 中某寄存器的内容直接交换
D. 只要有中断请求发生,那么一条指令执行结束后 CPU 就进入中断响应周期

37. 假设计算机系统中软盘以中断方式与 CPU 进行数据交换,主频为 50MHz,传输单位为 16 位,软盘的数据传输率为 50kB/s。若每次数据传输的开销(包括中断响应和中断处理)为 100 个时钟周期,则软盘工作时 CPU 用于软盘数据传输的时间占整个 CPU 时间的百分比是(　　)。
A. 0%　　B. 5%　　C. 1.5%　　D. 15%

38. 周期挪用方式常用于(　　)方式的输入/输出控制中。
A. DMA　　B. 中断　　C. 程序查询　　D. 通道

39. 采用"周期挪用"方式进行数据传送时,每传送一个数据要占用一个(　　)的时间。
A. 指令周期　　B. 机器周期　　C. 时钟周期　　D. 存储周期

40. DMA 方式的数据交换不是由 CPU 执行一段程序来完成,而是在(　　)之间建立一条逻辑上的直接数据通路,由 DMA 控制器来实现的。
A. CPU 与主存之间　　B. 外设与主存之间
C. 外设与 CPU 之间　　D. 外设与外设之间

41. 启动一次 DMA 传送,外设和主机之间将完成一个(　　)的数据传送。
A. 字节　　B. 字　　C. 总线宽度　　D. 数据块

42. 以下是有关对 DMA 方式的叙述:
① DMA 控制器向 CPU 请求的是总线使用权
② DMA 方式可用于键盘和鼠标器的数据输入
③ DMA 方式下整个 I/O 过程完全不需要 CPU 介入
④ DMA 方式需要用中断处理进行辅助操作
以上叙述中,错误的是(　　)。
A. ①、②　　B. ②、③　　C. ②、④　　D. ③、④

43. 以下关于 DMA 控制器和 CPU 关系的叙述中,错误的是(　　)。
A. DMA 控制器和 CPU 都可以作为总线的主控设备
B. DMA 控制器和 CPU 都要使用总线时,CPU 优先级更高
C. CPU 可通过执行 I/O 指令来访问 DMA 控制器中的寄存器
D. CPU 可通过执行 I/O 指令来启动进行 DMA 传送的设备

【参考答案】

1. A　2. D　3. D　4. A　5. D　6. D　7. A
8. D　9. D　10. A　11. D　12. C　13. A　14. C

15. C	16. A	17. D	18. D	19. A	20. C	21. B
22. C	23. C	24. C	25. D	26. A	27. B	28. D
29. A	30. A	31. A	32. C	33. A	34. C	35. A
36. D	37. B	38. A	39. D	40. B	41. D	42. B
43. B						

9.6 分析应用题

1. 对于磁盘来说,扇区的编号方式直接影响磁盘数据的读写时间。图 9.1 所示的两种扇区编号方式中,哪一种编号方式可能具有更好的性能?

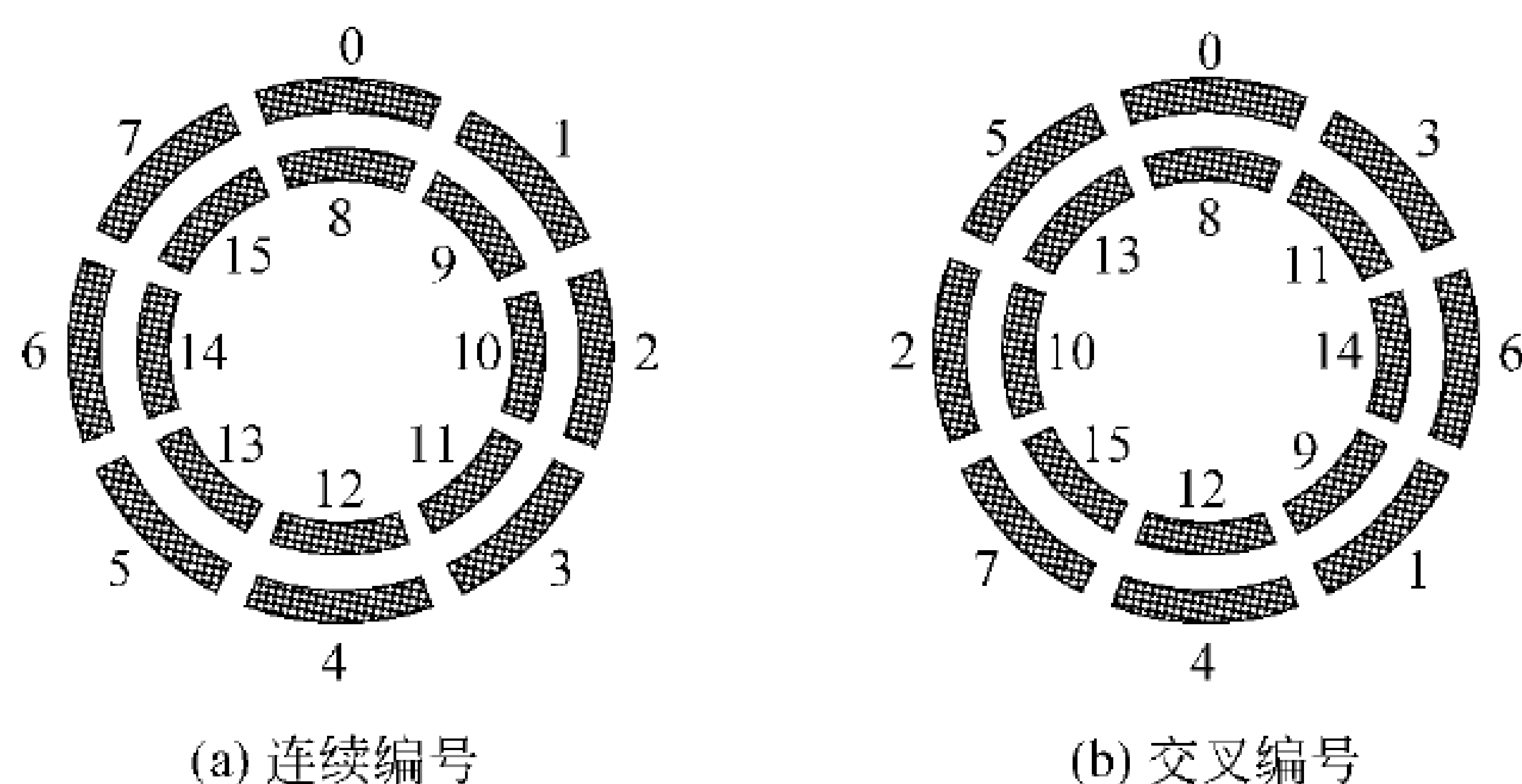


图 9.1 磁盘扇区编号方式

【分析解答】

交错因子是指每两个连续逻辑扇区之间所间隔的物理扇区数。显然,图 9.1(a)所示的交错因子是 0,图 9.1(b)所示的交错因子是 2。交错因子是硬盘低级格式化时,需要给定的一个主要参数,具体数值视硬盘类型而定。交错因子对硬盘的存取速度有很大影响。虽然硬盘的物理扇区在磁道上是连续排列的,但进行格式化后的逻辑扇区却是交叉排列的,也就是说,连续的物理扇区对应不连续的逻辑扇区。

硬盘每当访问一个逻辑扇区后,需等待主机将该扇区的输出数据处理完毕后才能进行下一个扇区的读写。在这个等待过程中,硬盘可能已经转过了几个物理扇区。如果交错因子选择过小,则对应下一个逻辑扇区的物理扇区已转过磁头,需等待磁盘再转一圈后才能读写;如果交错因子选取过大,则对应下一个逻辑扇区的物理扇区还未转到磁头处,需要继续等待。因此,选择合适的交错因子,可使当前扇区到下一个待读写的逻辑扇区之间没有或具有最短的等待时间,从而明显提高硬盘的读写速度。因此,图 9.1(b)中所示的交叉编号方式可能具有更好的性能。

2. 假定有一个磁盘存储器,磁盘片外径为 355.6mm,有 20 个记录面,每面有 51mm 区域用于记录信息,道密度为 3.92TPM(道/毫米),位密度为 90BPM(位/毫米),转速为 2400RPM,道间移动时间为 0.2ms。请问:

(1) 磁盘容量约为多少?如果道密度和位密度同时扩大 100 倍,则容量约为多少?

(2) 平均存取时间和平均数据传输率各为多少?

(3) 如果在道密度和位密度同时扩大 100 倍的同时转速扩大 3 倍,则平均存取时间和

平均数据传输率各为多少？

【分析解答】

(1) 每面磁道数为 51×3.92=200 道，给出的位密度是指最内圈上的位密度。所以，最内圈周长为 3.14×(355.6−2×51)≈796.3mm，故每道信息量为 796.3×90≈71664 位，因此，磁盘容量为 20×200×71664=286656000b≈273Mb(1M=2^{20})。若道密度和位密度同时扩大 100 倍，则磁道数扩大 100 倍，每道容量扩大 100 倍，所以整个盘组容量扩大 10000 倍，磁盘容量大约为 273Mb×10^4≈2666Gb≈333GB(1G=2^{30})。

(2) 平均寻道时间为(199×0.2+0)/2=19.9ms。转一圈时间为 60×10^3/2400=25ms，平均等待时间为(25+0)/2=12.5ms。平均存取时间为平均寻道时间与平均等待时间之和，即 19.9+12.5=32.4ms。平均数据传输率为 71664×10^3/25≈2.87Mb/s(1M=10^6)。

(3) 道密度扩大 100 倍，则平均寻道时间约为(19999×0.002+0)/2≈20ms。若转速扩大 3 倍，则转一圈的时间缩短为 25/3=8.33ms，平均等待时间缩短为 8.33/2=4.2ms，故平均存取时间为 20+4.2=24.2ms。位密度扩大 100 倍，则每道容量扩大 100 倍，转速扩大 3 倍，则转一圈的时间缩短为 1/3，因此，平均数据传输率共扩大 100×3=300 倍，即 300×2.87Mb/s=861Mb/s。

3. 假定一个程序重复完成以下将磁盘上一个 4KB 的数据块读出，进行相应处理后，写回到磁盘的另外一个数据区。各数据块内信息在磁盘上连续存放，数据块随机地位于磁盘的一个磁道上。磁盘转速为 7200RPM，平均寻道时间为 10ms，磁盘最大数据传输率为 40MBps，磁盘控制器的开销为 2ms，没有其他程序使用磁盘和处理器，并且磁盘读写操作和磁盘数据的处理时间不重叠。若程序对磁盘数据的处理需要 20000 个时钟周期，处理器时钟频率为 500MHz，则该程序完成一次数据块“读出-处理-写回”操作所需要的时间为多少？每秒钟可以完成多少次这样的数据块操作？

【分析解答】

磁盘转一圈的时间为 1000/7200≈8.33ms，故平均等待时间约为 8.33/2=4.17ms。数据块内信息连续存放，故数据块传输时间为 4×2^{10}/(40×10^6)=0.1024ms，因而数据块的平均读取或写回时间为 2ms+10ms+4.17ms+0.1024ms≈16.27ms。数据块的处理时间为 20000/500M=0.04ms。因为数据块随机存放在某个磁道上，所以，每个数据块的“读出-处理-写回”操作时间都是相同的，其整个时间为 16.27ms×2+0.04ms=32.58ms。所以每秒钟可以完成这样的数据块操作次数是 1000ms/32.58ms≈30 次。

4. 假定主存和磁盘存储器之间连接的同步总线具有以下特性：支持 4 字和 16 字两种长度(字长 32 位)的传送，总线时钟频率为 200MHz，总线宽度为 64 位，每个 64 位数据的传送需 1 个时钟周期，向主存发送一个地址需要 1 个时钟周期，每个总线事务之间有 2 个空闲时钟周期。若访问主存时最初 4 个字的存取时间为 200ns，随后每存取一个 4 字的时间是 20ns，磁盘的数据传输率为 5MB/s，则在 4 字和 16 字两种方式下，该总线上分别最多可有多少个磁盘同时进行传输？

【分析解答】

由第 8 章 8.6 节分析应用题 6 可知，在 4 字传输方式下，总线数据传输率为 71.11MB/s，因为 71.11/5=14.2，所以，该总线上最多可以有 14 个磁盘同时进行传输。在 16 字传输方式下，总线的数据传输率为 224.56MB/s，因为 224.56/5=44.9，因此，此时该总线上最多可

以有 44 个磁盘同时进行传输。

5. 假定有两个用来存储 10TB 数据的 RAID 系统。系统 A 使用 RAID1 技术，系统 B 使用 RAID5 技术。

(1) 系统 A 比系统 B 需要多用多少存储容量？

(2) 假定某个应用需要向磁盘写入一块数据，若磁盘读或写一块数据的时间为 30ms，则最坏情况下，在系统 A 和系统 B 上写入一块数据分别需要多长时间？

(3) 哪个系统更可靠？为什么？

【分析解答】

(1) 系统 A 使用 RAID1 技术，采用磁盘镜像方式存储，所以，所用磁盘容量为 10＋10＝20TB。系统B 使用 RAID5 技术，采用一个奇偶校验盘，假设使用 5 个磁盘阵列，那么 10TB 的数据需要 2.5TB 来存放冗余的奇偶校验数据，所以系统 A 比系统 B 多用 7.5TB 存储容量。

(2) 系统 A 的写入速度取决于两个磁盘中速度慢的那个，因为所有盘写一块数据的时间都是 30ms，故系统 A 写入一块数据的时间是 30ms。对于系统 B，最坏的情况下，写一块数据的时间为 2 次读和 2 次写，即所用时间为 $4\times30=120$ms。

(3) 系统 A 更可靠，因为系统对整个磁盘都进行了备份，所以即使所有的数据都损坏了也可以恢复，而系统 B 只是记录了部分冗余信息，如果两个磁盘的相同位都损坏了就恢复不出来了。

6. 假定在一个使用 RAID5 的系统中，采用先更新数据块、再更新校验块的信息更新方式。如果在更新数据块和更新校验块的操作之间发生了掉电现象，那么会出现什么问题？采用什么样的信息更新方式可避免这个问题？

【分析解答】

对于 RAID5 来说，如果在写完数据块但未写入校验块时发生断电，则写入的数据和对应的校验信息不匹配，无法正确恢复数据。这种情况可以避免，因为 RAID5 是大数据块交叉方式，每个盘独立进行操作，所以，只要同时写数据块所在盘和校验块所在盘即可。

7. 某终端通过 RS-232 串行通信接口与主机相连，采用起止式异步通信方式，若传输速率为 1200 波特，采用两相调制技术，通信协议为 8 位数据，无校验位，停止位为 1 位，请回答下列问题：

(1) 传送一个字符所需时间约为多少？

(2) 若传输速度为 2400 波特，停止位为 2 位，其他条件不变，则传输一个字符的时间约为多少？

(3) 若采用四相调制技术，其他条件不变，则传输一个字符的时间约为多少？

【分析解答】

(1) 采用两相调制技术，比特率＝波特率。1200 波特说明每秒钟传输 1200 个信息位。每个字符都有一个起始位，故一个字符有 1＋8＋1＝10 位，因而传输一个字符所需时间约为 $10\times(1/1200)\times10^3=8.3$ms。

(2) 一个字符有 11 位，传送一个字符的时间约为 $11\times(1/2400)\times10^3=4.6$ms。

(3) 采用四相调制技术，每个码元调制出 2 位信息，因而比特率为波特率的两倍，故传

输一个字符所需时间约为 $10\times(1/2400)\times10^3=4.15$ms。

8. 假定采用独立编址方式对 I/O 端口进行编号，那么，必须为处理器设计哪些指令来专门用于进行 I/O 端口的访问？连接处理器的总线必须提供哪些控制信号来表明访问的是 I/O 空间？

【分析解答】

若采用独立编址方式对 I/O 端口进行编号，则主存地址编号和 I/O 端口编号可能会相同，所以，无法利用访存指令来访问 I/O 端口，必须提供专门的输入/输出指令，包括 I/O 读指令和 I/O 写指令。在执行 I/O 指令时，CPU 会送出相应的 I/O 读和 I/O 写控制信号，以使和执行访存指令时送出的存储器读和存储器写信号有所区别。

9. 假设有一个磁盘，每面有 200 个磁道，盘面总存储容量为 1.6MB(为计算方便起见，设 $1M=10^6$)，磁盘旋转一周时间为 25ms，每道有 4 个数据区，每两个数据区之间有一个间隙，磁头通过每个间隙需 1.25ms。请回答下列问题：

(1) 从该磁盘上读取数据时的最大数据传输率是多少？

(2) 假如有人为该磁盘设计了一个与主机之间的接口，如图 9.2 所示，磁盘每读出一位，串行送入一个移位寄存器，每当移满 16 位后向处理器发出一个请求交换数据的信号。在处理器响应该请求信号并读取移位寄存器内容的同时，磁盘继续读出下一位数据并串行送入移位寄存器，如此继续工作。已知处理器在接到请求交换的信号以后，最长响应时间是 3 微秒，那么，这样设计的接口能否正确工作？若不能则应如何改进？

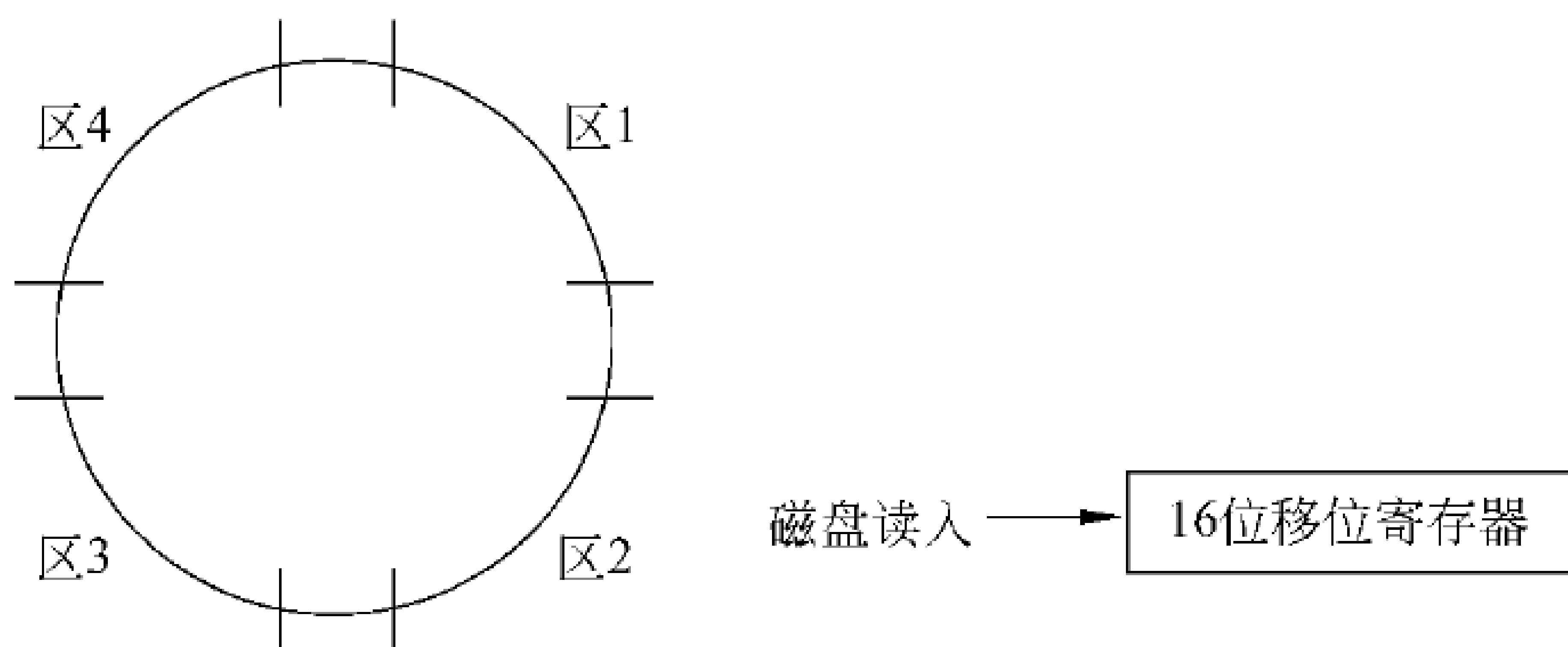


图 9.2　题 9 的示意图

【分析解答】

(1) 磁道容量为 $1.6\times10^6/200=8000$B，数据区容量为 $8000/4=2000$B，转过数据区的时间为 $(25-1.25\times4)/4=5$ms，故磁盘最大数据传输率为 2000B/5ms=0.4MB/s。

(2) 磁盘传送 1 位的时间为 $10^6/(0.4\times10^6\times8)=0.31\mu s<3\mu s$。因为传送 1 位的时间小于 3 微秒，所以，当处理器经过 $3\mu s$ 来读取移位寄存器中的数据时，磁盘已经读出了新的数据位，并将原先请求被读的在移位寄存器中的数据冲刷掉了。显然，这样的设计接口不能正确工作。磁盘读出一个字(16 位)所用时间为 $2/(0.4\times10^6)=5\mu s>3\mu s$，所以可以在磁盘接口中增加一个 16 位数据缓冲器。当 16 位移位寄存器装满后，首先送入数据缓冲寄存器，在读出下一个 16 位数据期间，上次读出的 16 位数据从数据缓冲器中被取走。

10. 假定一个政府机构同时监控 100 路移动电话的通话消息，通话消息被分时复用到一个带宽为 4MBps 的网络上，100 路复用使得每传送 1KB(1K=1024)的通话消息需额外开销 $150\mu s$，若通话消息的采样频率为 4kHz(1k=1000)，每个样本的量化值占 16 位。请回

答下列问题：

(1) 要求计算每个通话消息的传输时间，并判断该网络带宽能否支持同时监控100路通话消息？

(2) 若网络带宽降为1MBps，每次通话消息的额外开销为350μs，则该系统能否正确工作？

【分析解答】

(1) 该网络传输1KB的通话消息所需时间为 $150\mu s+10^6\times1024/(4\times10^6)=406\mu s$。所以，一秒钟内网络可传输的消息个数为 $10^6/406=2463$。每一路移动电话一秒钟所产生的数据量为 $4000\times2B=8000B=7.81KB$，所以，100路移动电话在一秒钟内所产生的消息个数为 $7.81\times100=781$，因为 $781<2463$，故该网络带宽可支持同时监控100路通话消息。

(2) 每一路移动电话一秒钟所需传输的数据量为 $4000\times2B=8000B=7.81KB$，100个移动电话在一秒钟内产生的数据量为 $100\times7.81KB=781KB$，所以，每秒钟共产生781个消息。如果不考虑额外开销带来的延迟，那么，每秒钟产生781KB的信息量在带宽为1MB/s的网络上传输是没有问题的。

但是，由于多路复用带来了额外开销，使得该网络传输每个1KB的通话消息所需时间为 $350+10^6\times1024/(1\times10^6)=1374\mu s=1.374ms$。所以1秒钟内网络上能够传输的消息最多只有 $1000ms/1.374ms=740$ 个，而100个移动电话在一秒钟内共产生了781个消息（所需时间为 $781\times1.374ms\approx1070ms>1$ 秒），所以系统不能正确工作。

11. 假定一台计算机带有20个终端同时工作，在运行用户程序的同时，能接收来自任意一个终端输入的字符信息，并将字符回送显示或打印。每一个终端的键盘输入部分有一个数据缓冲寄存器 $RDBR_i$（$i=1\sim20$），当在键盘上按下某一个键时，相应的字符代码即进入 $RDBR_i$，并使它的“完成”状态标志 $Done_i$（$i=1\sim20$）置1，要等处理器把该字符代码取走后，$Done_i$ 标志才自动清“0”（复位）。每个终端显示或打印输出部分也有一个数据缓冲寄存器 $TDBR_i$（$i=1\sim20$），并有一个 $Ready_i$（$i=1\sim20$）状态标志，该状态标志为1时，表示相应的 $TDBR_i$ 是空着的，准备接收新的输出字符代码，当 $TDBR_i$ 接收了一个字符代码后，$Ready_i$ 标志自动清“0”，并将字符送终端显示或打印。为了接收终端的输入信息，处理器为每个终端设计了一个指针 PTR_i（$i=1\sim20$）指向为该终端保留的主存输入缓冲区。处理器采用下列两种方案输入键盘代码，同时回送显示或打印。

(1) 每隔一固定时间T转入一个状态检查程序DEVCHC，顺序地检查全部终端是否有任何键盘信息输入，如果有，则顺序完成。

(2) 允许任何有键盘信息输入的终端向处理器发出中断请求。全部终端采用共同的向量地址，利用它使处理器在响应中断后，转入一个中断服务程序DEVINT，由后者询问各终端状态标志，并为最先遇到的有中断请求的终端服务，服务结束后返回用户程序。

要求画出DEVCHC和DEVINT两个程序的流程图。

【分析解答】

定时查询程序DEVCHC和中断服务程序DEVINT的流程分别如图9.3和图9.4所示。图中用(x)表示寄存器x或存储单元x的内容。x可能是存储单元地址或寄存器编号。此外，因为标志 $Done_i$ 和 $Ready_i$ 由硬件控制自动清“0”（复位），所以，流程图中不需要有对这两个标志赋值的操作。程序DEVINT的流程图中，如果所有终端都检测不到Done标志为1，说明所有终端都没有键盘输入，即都没有中断请求，此时不应该进入DEVINT处理，

因此需报告“出错”。

① 程序 DEVCHC

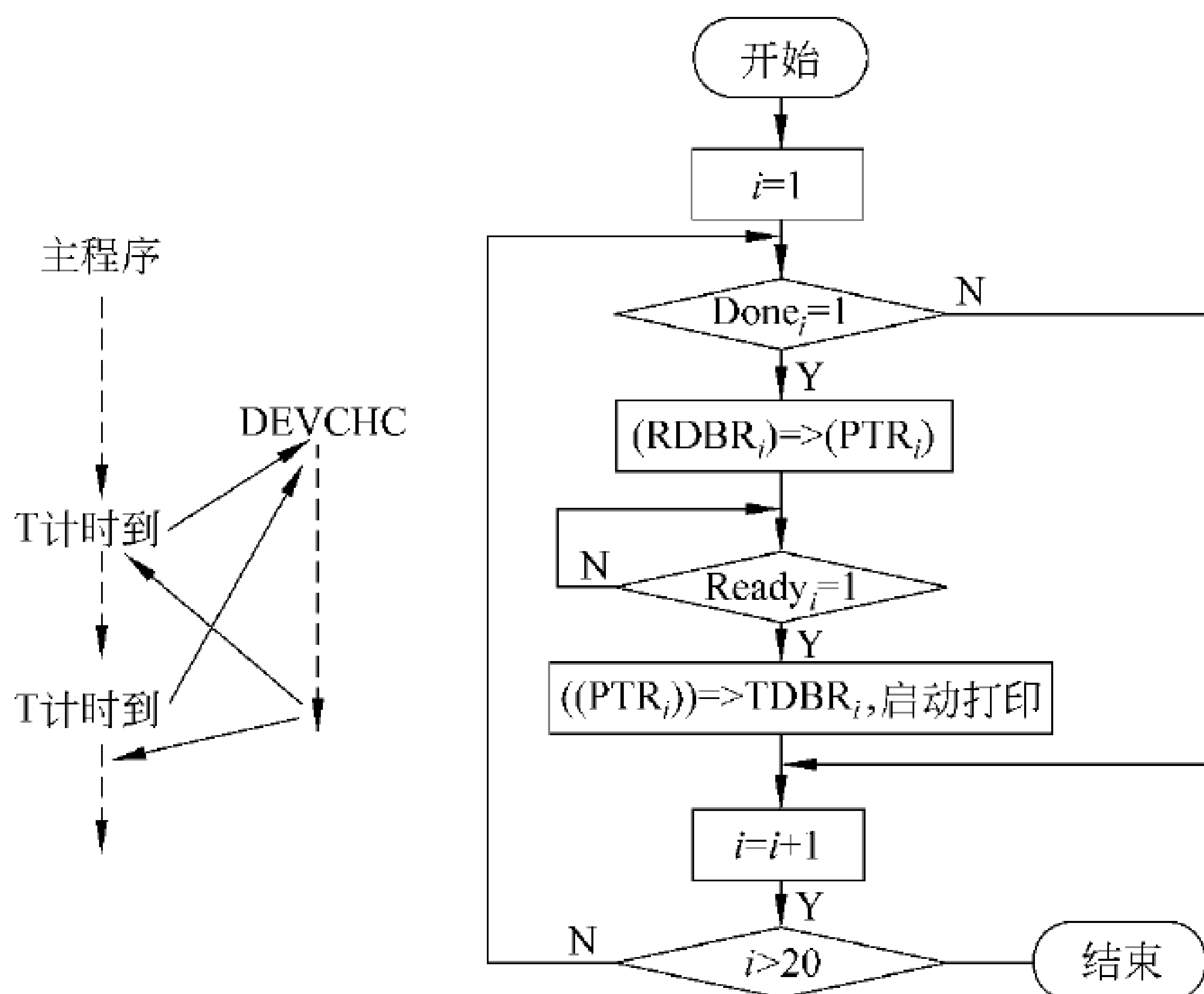


图 9.3 定时查询程序 DEVCHC 的处理流程

② 程序 DEVINT

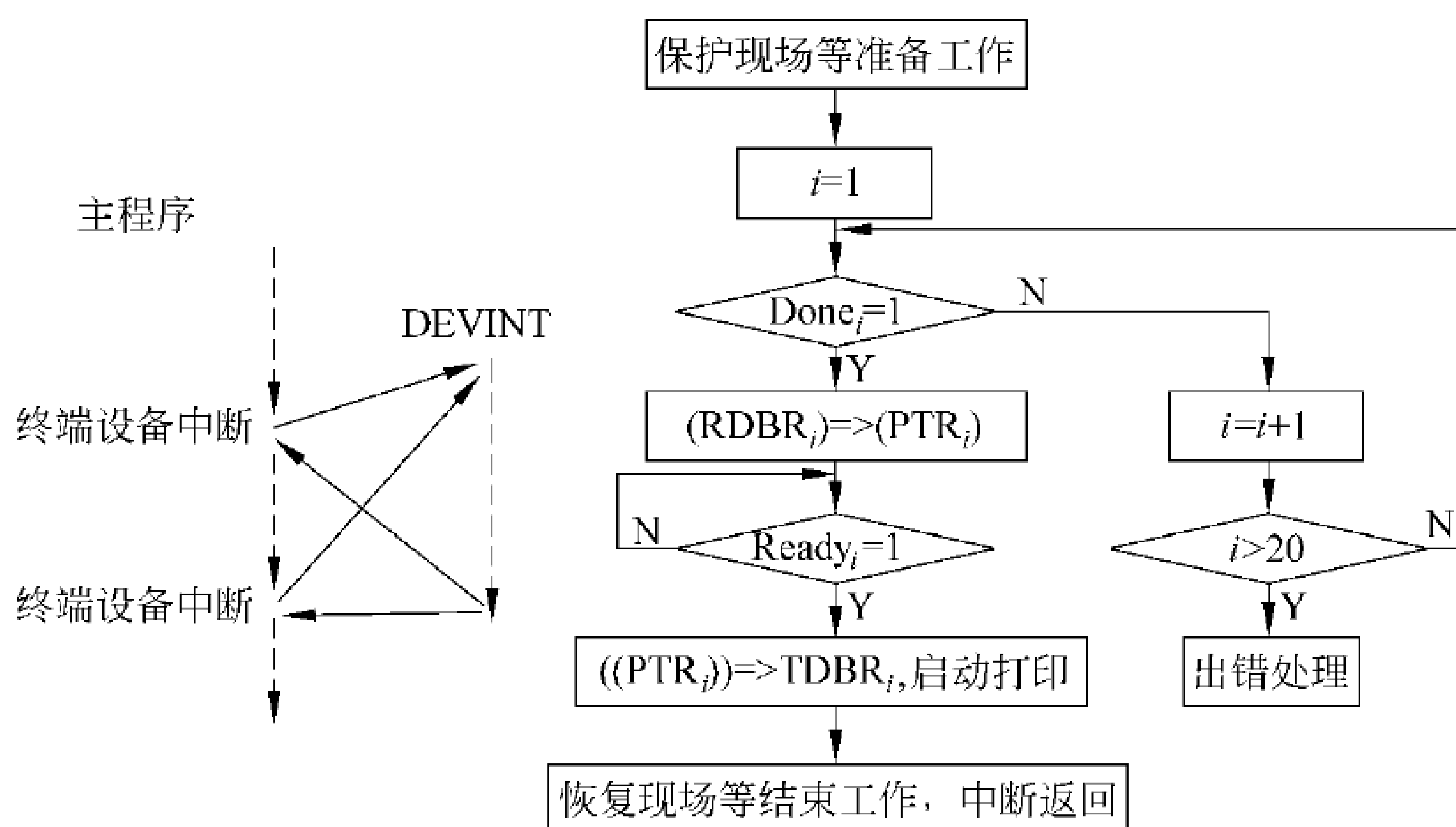


图 9.4 中断服务程序 DEVINT 的处理流程

12. 若某计算机有 5 级中断，中断响应优先级为 1>2>3>4>5，而中断处理优先级为 1>4>5>2>3。要求：

(1) 设计各级中断服务程序的中断屏蔽位(假设 1 为屏蔽，0 为开放)。

(2) 若在运行用户程序时，同时出现第 2、4 级中断请求，而在处理第 2 级中断过程中，又同时出现 1、3、5 级中断请求，试画出此时 CPU 运行过程示意图。

【分析解答】

(1) 由题意可知，1 级中断的处理优先级最高，说明 1 级中断对其他所有中断都屏蔽，其屏蔽字为全 1；3 级中断的处理优先级最低，所以除了 3 级中断本身之外，对其他中断全都开放，其屏蔽字为 00100。以此类推，得到所有各级中断的中断服务程序中设置的中断屏蔽字

如表 9.1 所示。

表 9.1 题 12 的中断屏蔽字

中断处理程序	中断屏蔽字				
	第 1 级	第 2 级	第 3 级	第 4 级	第 5 级
第 1 级	1	1	1	1	1
第 2 级	0	1	1	0	0
第 3 级	0	0	1	0	0
第 4 级	0	1	1	1	1
第 5 级	0	1	1	0	1

(2) 在运行用户程序时，同时出现 2、4 级中断请求，因为用户程序对所有中断都开放，所以，在中断响应优先级排队电路中，有 2、4 两级中断进行排队判优，根据中断响应优先级 2>4，因此先响应 2 级中断。在 CPU 执行 2 级中断服务程序过程中，首先保护现场、保护旧屏蔽字、设置新的屏蔽字 01100，然后，在具体中断处理前先开中断。一旦开中断，则马上响应 4 级中断，因为 2 级中断屏蔽字中对 4 级中断的屏蔽位是 0，即对 4 级中断是开放的。在执行 4 级中断结束后，回到 2 级中断服务程序执行；在具体处理 2 级中断过程中，同时发生了 1、3、5 级中断请求，因为2 级中断对 1、5 级中断开放，对 3 级中断屏蔽，所以只有 1 和 5 两级中断进行排队判优，根据中断响应优先级 1>5，所以先响应 1 级中断。因为 1 级中断处理优先级最高，所以在其处理过程中不会响应任何新的中断请求，直到 1 级中断处理结束，然后返回 2 级中断；因为 2 级中断对 5 级中断开放，所以在 2 级中断服务程序中执行一条指令后，又转去执行 5 级中断服务程序，执行完后回到 2 级中断，在 2 级中断服务程序执行过程中，虽然 3 级中断有请求，但是，因为 2 级中断对 3 级中断不开放，所以，3 级中断一直得不到响应。直到 2 级中断处理完回到用户程序，才能响应并处理 3 级中断。CPU 运行程序的执行过程如图 9.5 所示。

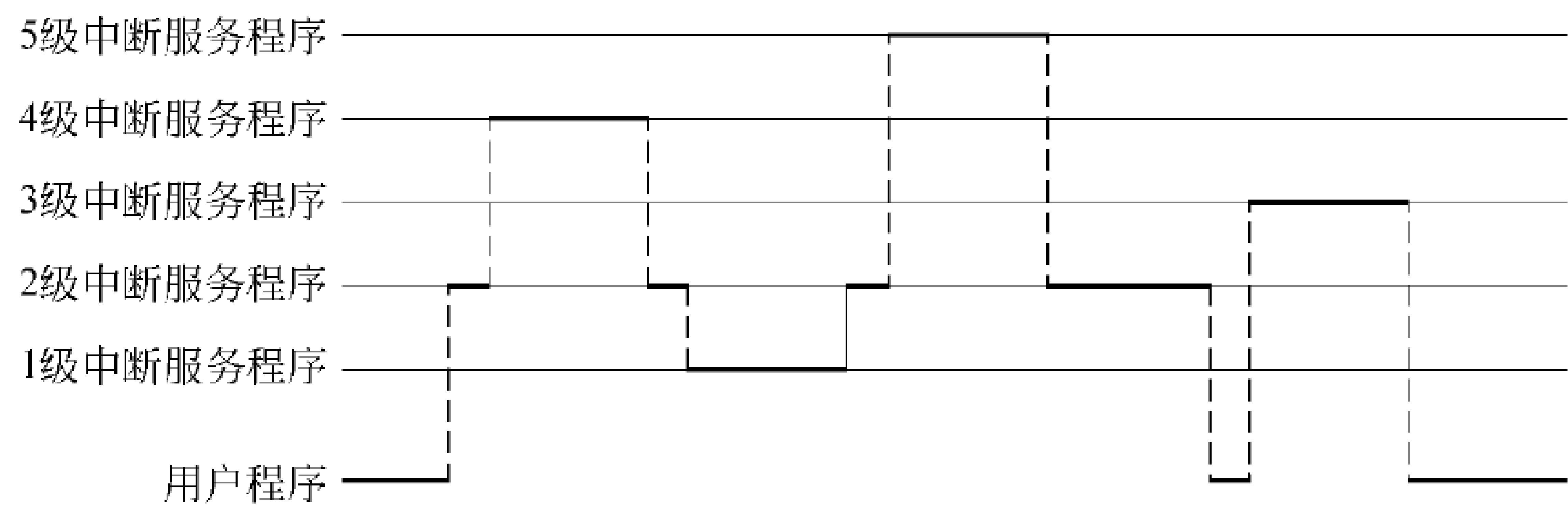


图 9.5 CPU 运行程序的执行过程

13. 某计算机的 CPU 主频为 500MHz，所连接的某外设的最大数据传输率为 20kBps，该外设接口中有一个 16 位的数据缓存器，相应的中断服务程序的执行时间为 500 个时钟周期。请回答下列问题：

(1) 是否可用中断方式进行该外设的输入输出？若能，在该设备持续工作期间，CPU 用于该设备进行输入/输出的时间占整个 CPU 时间的百分比大约为多少？

(2) 若该外设的最大数据传输率是 2MBps，则可否用中断方式进行输入输出？

【分析解答】

(1) 因为该外设接口中有一个16位数据缓存器，所以，若用中断方式进行输入/输出，可以每16位进行一次中断请求，因此，中断请求的时间间隔为$10^6 \times 2B/20kB = 100\mu s$。

对应的中断服务程序的执行时间为$(10^6/500M) \times 500 = 1\mu s$，因为中断响应过程就是执行一条隐指令的过程，所用时间相对于中断处理时间(执行中断服务程序的时间)而言，几乎可以忽略不计，因而整个中断响应并处理的时间大约为$1\mu s$多一点，远远小于中断请求的间隔时间。因此，可以用中断方式进行该外设的输入输出。

若用中断方式进行该设备的输入/输出，则该设备持续工作期间，CPU用于该设备进行输入/输出的时间占整个CPU时间的百分比大约为$1/100 = 1\%$。

也可以通过考察1秒钟内500M个时钟周期中有多少时钟周期用于中断来计算百分比，其结果为$(10^6/100 \times 500)/500M = 1\%$。

(2) 若外设的最大传输率为2MBps，则中断请求的时间间隔为$10^6 \times 2B/2MB = 1\mu s$。而整个中断响应并处理的时间大约为$1\mu s$多一点，中断请求的间隔时间小于中断响应和处理时间，即中断处理还未结束就会有该外设新的中断到来，所以不可以用中断方式进行该外设的输入输出。

14. 假设某计算机中软盘以中断方式进行数据输入/输出，每次中断请求传输一个32位数。已知软盘的数据传输率为500kB/s，每次传输的CPU开销(包括中断响应和处理)为1000个时钟周期，CPU的主频为500MHz，则软盘在持续工作时，CPU用于软盘数据传送的时间占CPU整个时间的百分比是多少?

【分析解答】

软盘准备32位数据的时间为$10^6 \times 4B/500kB = 8\mu s$。因此，软盘每隔8微秒发一次中断请求，CPU响应并处理中断所用时间为$1000 \times 10^6/(500 \times 10^6) = 2\mu s$，因此，每次CPU花$2\mu s$取走数据后，就去执行其他程序；过$8\mu s$后软盘又准备好下一个数据，又发中断请求，CPU响应并处理中断以取走数据，然后又去执行其他程序；……，如此周而复始，直到所有需要的数据传送完。因此，当软盘持续工作时，CPU用于软盘数据传送的时间占CPU总时间的百分比是$2/8 = 0.25 = 25\%$。

15. 某计算机CPU主频为500MHz，CPI为5。假定某外设的数据传输率为0.5MB/s，采用中断方式与主机进行数据传送，传输单位为32位，对应的中断服务程序包含18条指令，中断响应等其他开销相当于两条指令的执行时间。请回答下列问题，要求给出计算过程。

(1) 在中断方式下，CPU用于该外设I/O的时间占整个CPU时间的百分比是多少?

(2) 当该外设的数据传输率达到5MB/s时，改用DMA方式传送数据。假定每次DMA传送的块大小为5000B，DMA预处理和后处理的总开销为500个时钟周期，则CPU用于该外设I/O的时间占整个CPU时间的百分比是多少?(假设DMA与CPU之间没有访存冲突)

【分析解答】

(1) 中断方式下，每当外设准备好32位数据(读操作)或外设接口中的32位数据缓存为空已准备好接收新数据(写操作)时，就向CPU发中断申请，要求CPU通过执行中断服务程序来取走缓存中的32位数据或向缓存送32位数据。

CPU每次执行中断服务程序花18条指令的时间，其他如中断响应等的开销相当于

两条指令的时间，CPI 为 5。因此，每次 CPU 用于中断处理(数据传送服务)的时钟周期数为5×18+5×2=100。

外设的数据传输率为 0.5 MB/s，每次中断传送 32 位数据，占 4 个字节，因此，外设每秒钟申请的中断次数为 0.5MB/4B=125000，因而每秒钟内 CPU 用于中断响应和处理的时间开销为 100×125000=12500000=12.5M 个时钟周期，CPU 的时钟频率为 500MHz，即 CPU 每秒钟内产生 500M 个时钟周期，故 CPU 用于外设 I/O 的时间占整个 CPU 时间的百分比为 12.5M/500M=2.5%(也可通过考察相邻两次中断请求间隔时间内 CPU 用于中断的时间来计算，即(100×1/500M)/(4B/0.5MB)=2.5%)。

(2) 当外设数据传输率提高到 5MB/s 时，1 秒钟内产生的 DMA 次数为 5MB/5000B=1000；每次 DMA 传送前都需要进行 DMA 初始化(预处理)，DMA 结束后还要进行中断处理(后处理)，已知这两个处理总共需要 500 个时钟周期，所以，CPU 用于 DMA 处理的总开销为 1000×500=500000=0.5M 个时钟周期；而 CPU 的时钟频率为 500MHz，即 CPU 每秒钟内产生 500M 个时钟周期，故 CPU 用于该外设 I/O 的时间占整个 CPU 时间的百分比为 0.5M/500M=0.1%(也可通过考察相邻两次 DMA 请求间隔时间内 CPU 用于该外设 I/O 的时间来计算，即(500×1/500M)/(5000B/5MB)=0.1%)。

16. 假设某计算机字长 16 位，没有 cache，运算器一次定点加法时间等于 100 毫微秒，配置的磁盘旋转速度为每分钟 3000 转，每个磁道上记录两个数据块，每一块有 8000 个字节，两个数据块之间间隙的越过时间为 2 毫秒，主存存储周期为 500 毫微秒，存储器总线宽度为 16 位。

(1) 磁盘读写数据时的最大数据传输率是多少？平均数据传输率是多少？

(2) 若磁盘按最大数据传输率与主存交换数据时 CPU 没有访问主存，则此时主存频带空闲百分比是多少？(主存频带空闲百分比指无数据读写的空闲存储周期数占所有存储周期总数的百分比)

(3) 直接寻址的“存储器-存储器”SS 型加法指令在无磁盘 I/O 操作打扰时的执行时间为多少？此时，主存频带空闲百分比是多少？当磁盘 I/O 操作与一连串这种 SS 型加法指令执行同时进行时，这种 SS 型加法指令的最快和最慢执行时间各是多少？(假定采用多周期处理器方式，CPU 时钟周期等于主存周期)

【分析解答】

(1) 磁盘旋转一周所需时间为 $60\times10^3/3000=20$ms，单个数据块的传输时间为(20ms/2)−2ms=8ms，所以最大数据传输率为 8000B/8ms=1MBps。平均数据传输率为 2×8000B/20ms=0.8MBps。

(2) 因为磁盘最大数据传输率为 1MBps，存储器总线宽度为 16bit=2B，所以，每隔 $10^9\times2$B/1MB=2000ns 产生一个 DMA 请求，即每 2000ns/500ns=4 个主存周期中有一个被 DMA 挪用，此时，CPU 没有访问主存，因此，4 个主存周期中有 3 个空闲，故主存频带空闲百分比是 75%，如图 9.6 所示。图中箭头处开始的一个主存周期被 DMA 挪用。

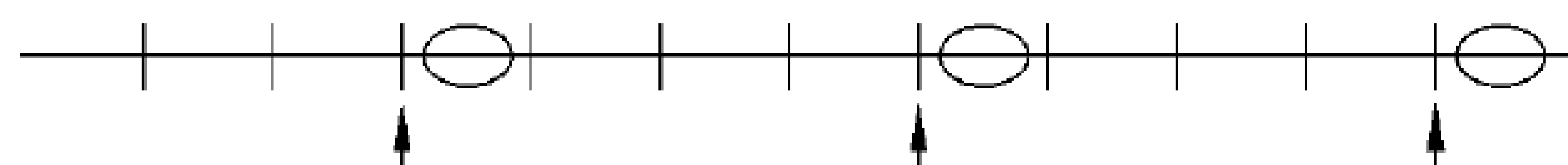


图 9.6 无 CPU 访存时主存周期被 DMA 使用的情况

(3) 无磁盘 I/O 打扰时，执行一条直接寻址的 SS 型加法指令时主存被 CPU 访问的过程如图 9.7 所示。该指令执行过程包括取指令、取源操作数 1、取目操作数(源操作数 2)、执行、写结果，共需 5 个时钟周期，因此执行时间为 5×500ns＝2.5μs。此时，每个指令周期所包含的 5 个时钟周期中，只有执行阶段不访问主存，所以主存频带空闲百分比是 20%。(图中箭头处即一条指令的第一个时钟周期开始)

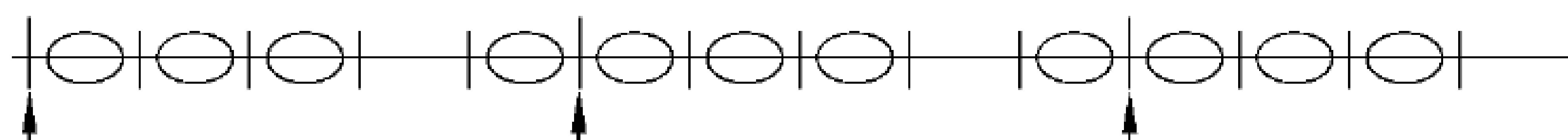


图 9.7　无磁盘 I/O 打扰时主存周期被 CPU 使用的情况

当磁盘 I/O 操作与一连串这种 SS 型加法指令同时进行时，可能因为 CPU 和 DMA 同时访存而使指令的执行时间被延长。每次 DMA 请求要求挪用一个主存周期来访问主存，同时，CPU 执行指令时也要求访问主存，当两者发生冲突时，DMA 优先级高，CPU 的访存请求被延迟响应。因为每隔 2000ns 产生一个 DMA 请求，因此每 4 个主存周期必定有一个被 DMA 所挪用。此时，主存周期的占用情况如图 9.8 所示。

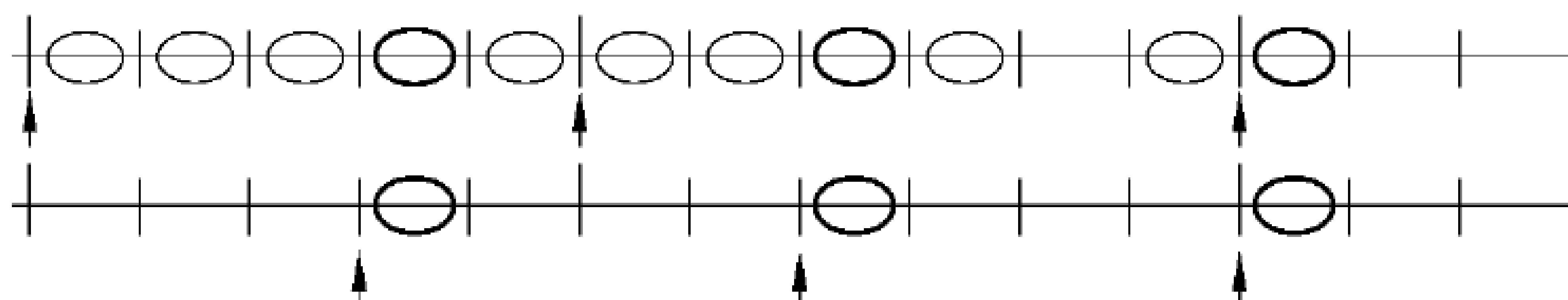


图 9.8　CPU 和 DMA 交替串行访问主存时的情况

由图 9.8 可知，最好的情况是在 SS 型加法指令执行过程中没有访存冲突(如图 9.8 中最开始的一个指令周期)，此时最快，指令执行时间为 2.5μs；最坏的情况是有一次访存冲突(如图 9.8 中第二个指令周期)，此时最慢，指令执行时间为 2.5μs＋500ns＝3μs。

17. 假设某计算机所有指令都可用两个总线周期完成，一个总线周期用来取指令，另一个总线周期用来存取数据。假定总线宽度为 8 位，每个总线周期为 250ns，因而每条指令的执行时间为 500ns。若该计算机中配置的磁盘每个磁道有 16 个 512 字节的扇区，磁盘旋转一圈的时间是 8.192ms。请回答下列问题。

(1) 在磁盘不工作时，主存频带空闲百分比是多少?

(2) 若采用周期挪用法进行 DMA 传送，则该计算机执行指令的速度由于 DMA 传送而降低了多少?

(3) 若采用周期挪用法进行 DMA 传送，总线宽度改为 16 位，则该计算机执行指令的速度由于 DMA 传送而降低了多少?

【分析解答】

(1) 因为所有指令的每个阶段都要访问主存，所以即使没有磁盘访问主存，CPU 也把主存周期占满了。因此，主存频带空闲百分比是 0。

(2) 磁盘的平均数据传输率为 10^3 ×16×512B/8.192＝1MBps。当总线位宽为 8 位时，DMA 控制器每隔 1B/1MB＝1μs 申请一次数据传送，在 1μs 期间 CPU 共执行 1μs/500ns＝2 条指令。因此，每两条指令的执行被插入一个总线周期用于一次数据传送，即平均每条指令延长了 250/2＝125ns。因而，计算机执行指令的速度降低了 125/500＝25%。

(3) 当总线位宽为 16 位时，DMA 控制器每隔 2B/1MB＝2μs 申请一次数据传送，在

$2\mu s$ 期间 CPU 共执行 $2\mu s/500ns=4$ 条指令，因此，每 4 条指令的执行被插入一个总线周期用于一次数据传送，即平均每条指令延长了 250/4=62.5ns。因而，计算机执行指令的速度降低了 62.5/500=12.5%。

18. 假设一个主频为 1GHz、CPI 为 5 的 CPU 需要从某个成块传送的 I/O 设备读取 1000 字节的数据到主存缓冲区中，该 I/O 设备一旦启动即按 50kBps 的数据传输率向主机传送 1000 字节数据，每个字节的读取、处理并存入内存缓冲区需要 1000 个时钟周期，则以下 4 种方式下，在 1000 字节的读取过程中，CPU 用在该设备的 I/O 操作上的时间分别为多少？占整个 CPU 时间的百分比分别是多少？

(1) 采用定时查询方式，每次处理一个字节，一次状态查询至少需要 60 个时钟周期。

(2) 采用独占查询方式，每次处理一个字节，一次状态查询至少需要 60 个时钟周期。

(3) 采用中断 I/O 方式，外设每准备好一个字节发送一次中断请求。每次中断响应需要 2 个时钟周期，中断服务程序的执行需要 1200 个时钟周期。

(4) 采用周期挪用 DMA 方式，每挪用一次主存周期处理一个字节，一次 DMA 传送完成 1000 字节的数据传送，DMA 初始化和后处理的时间为 2000 个时钟周期，CPU 和 DMA 之间没有访存冲突。

(5) 如果设备的速度提高到 5MBps，则上述 4 种方式中，哪些是不可行的？为什么？对于可行的方式，计算出 CPU 在该设备 I/O 操作上所用的时间占整个 CPU 时间的百分比。

(6) 如果外设不是成块传送设备，而是字符型设备，CPU 每处理完一个字节后都要重新启动外设，外设在启动后 0.02ms 时间内准备好一个字节。每个字节的读取、处理(包括启动下一次操作)并存入内存缓冲区还是需要 1000 个时钟周期，假定 CPU 总是在查询到就绪后立即启动外设或在中断服务程序执行了 20 条指令后启动外设，则在(1)～(3)三种方式下，CPU 在该设备的 I/O 操作上所用的时间占整个 CPU 时间的百分比分别是多少？

(7) 对以上各种情况进行分析后，你可以得出哪些结论？

【分析解答】

主频为 1GHz，所以，时钟周期为 1/1GHz=1ns。因为每个字节的读取、处理并存入内存缓冲区需要 1000 个时钟周期，所以，对于像程序查询和中断等用软件实现输入/输出的方式，CPU 为每个字节传送所用的时间至少为 $1000\times 1ns=1\mu s$。在 50kBps 的数据传输率下，设备每隔 $(10^6\times 1B/50kB)=20\mu s=20000ns$ 准备好一个字节，因而 1000 字节的传输时间为 $1000\times 20\mu s=20ms$。

(1) 定时查询方式下的 I/O 过程如图 9.9 所示。可以设置每隔 20000ns 查询一次，这样使得查询程序的开销达到最小，即第一次读取状态时就可能发现就绪，然后用 1000 个时钟周期进行相应处理，因此，对于每个字节的传送，CPU 所用时钟周期数为 60+1000=1060。因此，在 1000 个字节的读取过程中，CPU 用在该设备的 I/O 操作上的时间至少为 $1000\times 1060\times 1ns=1.060ms$，占整个 CPU 时间的百分比至少为 1.060/20=5.3%。

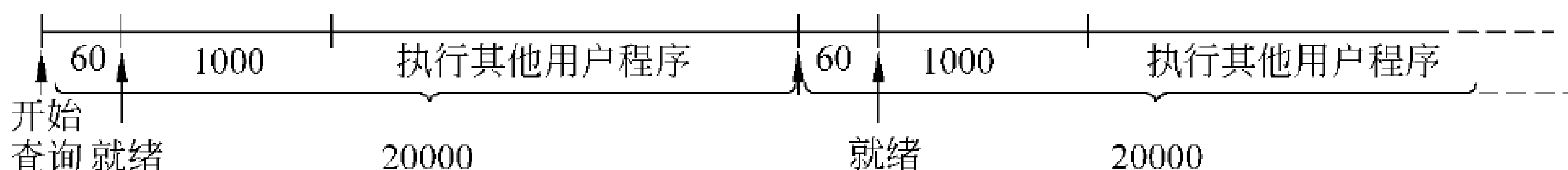


图 9.9 定时查询方式下的 I/O 过程

(2) 独占查询方式下的I/O过程如图9.10所示。启动设备后，CPU就开始查询，因为333×60+20=20000，所以第一个字节传送在第334次读取状态时检测到就绪，随后用1000个时钟周期进行相应的处理，然后继续第2个字节的状态查询。因为40+1000+316×60=20000，所以，第2个字节的传送在第316次读取状态时检测到就绪，第1和第2个字节的传送过程如图9.10(a)所示。每次检测到就绪后，就进行相应的处理，然后周而复始地进行查询，因为(20000−1000)/60=316.7，所以，第317次读取状态时发现就绪。因为1000+60×317−20000=20，所以，每3个字节可多60个时钟周期，正好进行一次状态查询，因此，在剩下的998个字节的读取过程中，前996个字节的传送正好用了996×20000个时钟周期，如图9.10(b)所示。最后两个字节的传送过程如图9.10(c)所示，因为2×(1000+60×317−20000)=40，此外，最后一个字节的处理还需1000个时钟周期，所以最后两个字节总的时间为2×20000+40+1000=41040个时钟周期。综上所述，CPU用在该设备的I/O操作上的总时间为(1000×20000+1040)×1ns=20.00104ms≈20ms。即在1000字节的整个传输过程中，CPU一直为该设备服务，所用时间占整个CPU时间的100%。

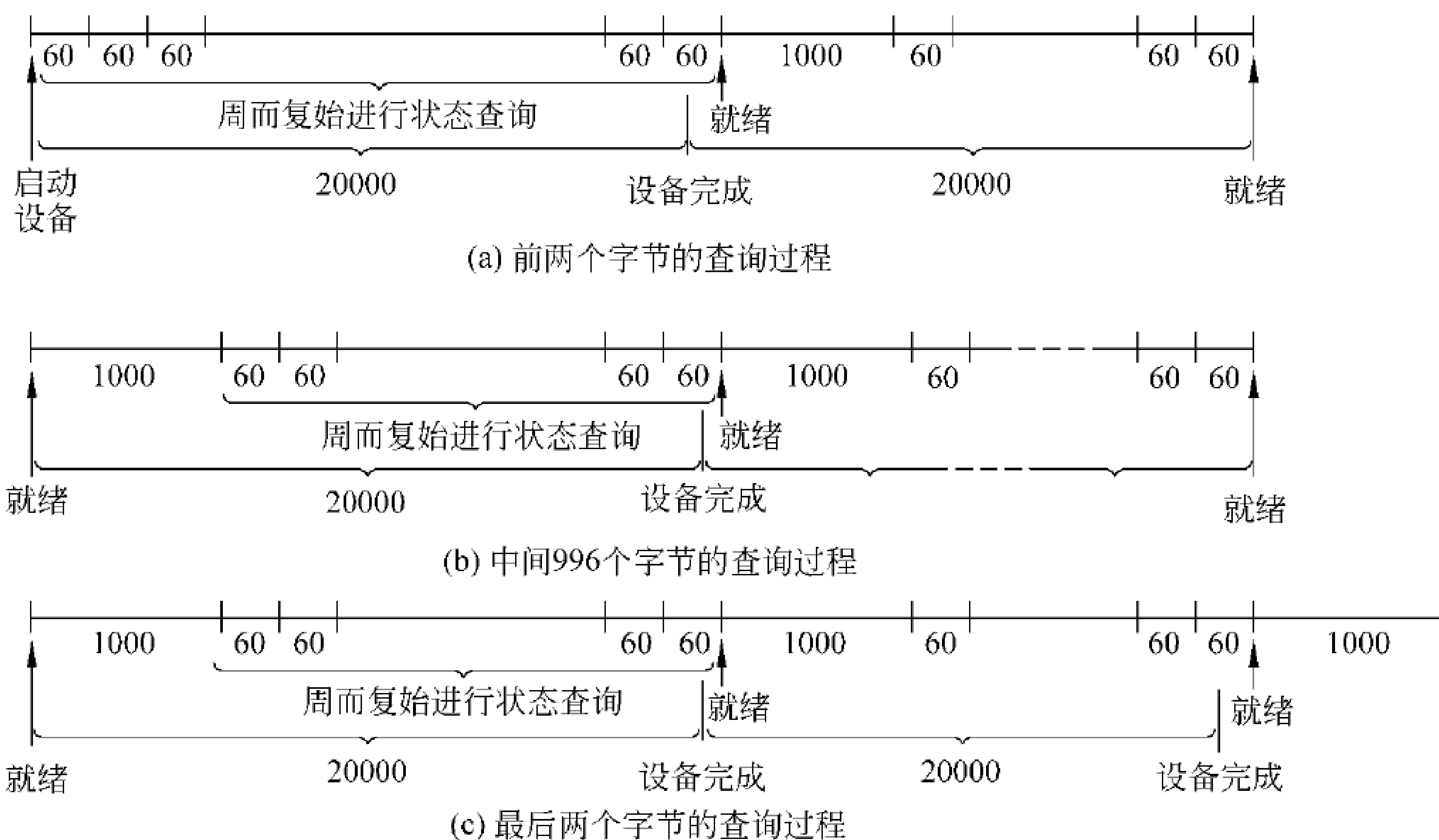


图9.10　独占查询方式下的I/O过程

(3) 中断方式下的I/O过程如图9.11所示。中断方式下，外设每准备好一个字节请求一次中断，每次中断CPU所用时钟周期数为2+1200=1202，因此CPU用在该设备的I/O操作上的时间为1000×1202×1ns=1.202ms，占整个CPU时间的百分比至少为1.202/20=6.01%。

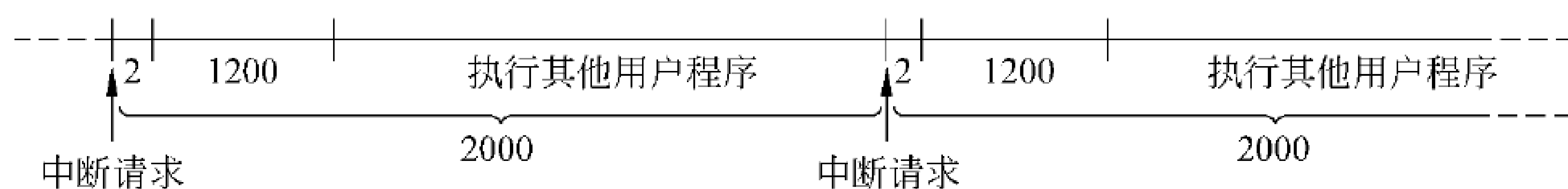


图9.11　中断方式下的I/O过程

(4) DMA 方式下，由于 CPU 和 DMA 没有访存冲突，所以不需考虑由于 DMA 而影响到 CPU 执行其他程序。因此，传送 1000 个字节 CPU 所用的时钟周期数就是 2000，在 1000 个字节的读取过程中，CPU 用在该设备的 I/O 操作上的时间为 $2000 \times 1\text{ns} = 2\mu s$，占整个 CPU 时间的百分比为 $2/(1000 \times 20) = 0.01\%$。

(5) 若数据传输率为 5MBps，则 1000 字节的传输时间为 $10^6 \times 1000\text{B}/5\text{MB} = 200\mu s$。

对于定时查询和独占查询方式，传送 1000 字节 CPU 所用时间至少为 $1000 \times (60 + 1000) \times 1\text{ns} = 1060\mu s$；对于中断方式，传送 1000 字节 CPU 所用时间为 $1000 \times (2 + 1200) \times 1\text{ns} = 1202\mu s$。上述 3 种方式下，CPU 所用的时间都比设备所用的传输时间长得多，即设备的传输比 CPU 的处理快得多，因而发生数据丢失。因此，这 3 种方式都不能用于该设备的 I/O 操作。对于 DMA 方式，传送 1000 字节 CPU 所用时间为 $2000 \times 1\text{ns} = 2\mu s$，占整个 CPU 时间的百分比为 $2/200 = 1\%$，说明可以使用 DMA 方式，不过由于外设传输速度加快，使得 CPU 频繁进行 DMA 预处理和后处理，因而 CPU 的开销从 0.01%上升到了 1%。

(6) 对于字符型设备，在定时查询方式下，其数据传送过程如图 9.12 所示。CPU 可以每隔 0.02ms(相当于 20000 个时钟周期)查询一次，这样总是在第一次查询时就发现就绪，马上启动设备进行下一个字节的传送，并读取和处理当前字节。因此，传送 1000 个字节 CPU 所用时间至少为 $1000 \times (1000 + 60) \times 1\text{ns} = 1060\mu s = 1.06\text{ms}$，而设备传输 1000 字节所用时间为 $0.02\text{ms} \times 1000 = 20\text{ms}$，因此占用 CPU 时间的百分比至少为 $1.06/20 = 5.3\%$。

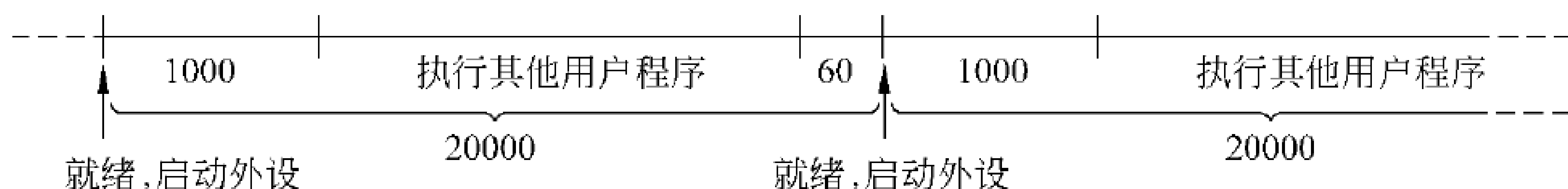


图 9.12 字符型设备定时查询方式下的 I/O 过程

在独占查询方式下，其数据传送过程如图 9.13 所示。第一个字节传送在第 334 次读取状态时检测到就绪，随后启动外设；在外设工作的同时，查询程序用 1000 个时钟周期进行相应的处理，然后周而复始地进行查询，因为 $20000 - 1000/60 = 316.7$，所以，对于后面的 999 个字节，每当进行到第 317 次状态查询时发现就绪。对于最后一个字节，外设就绪后 CPU 还需要 1000 个时钟对其进行处理。因此，在 1000 个字节的读取过程中，CPU 用在该设备 I/O 操作的时间为 $(334 \times 60 + 999 \times (1000 + 60 \times 317) + 999 \times 1000) \times 1\text{ns} = 20.02102\text{ms} \approx 20\text{ms}$。这种情况下，该设备的 I/O 操作占用 CPU 的时间为 100%。

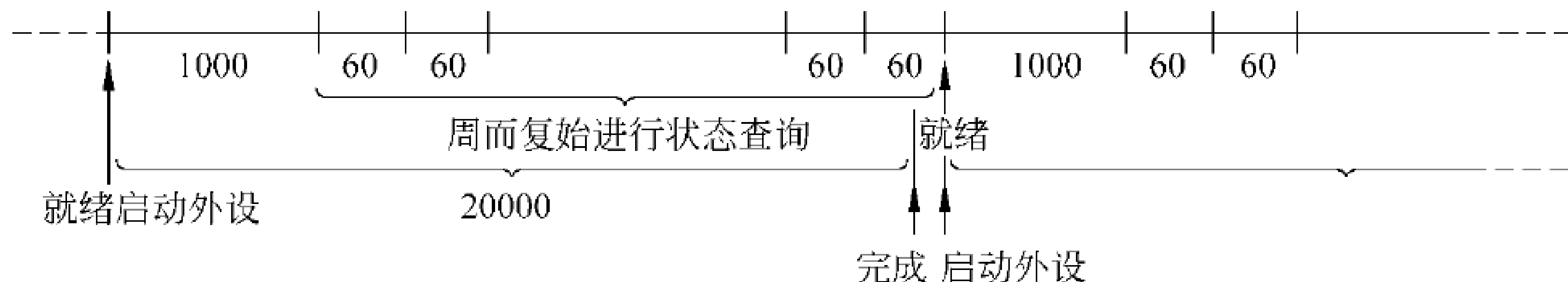


图 9.13 字符型设备独占查询方式下的 I/O 过程

在中断方式下，其数据传送过程如图 9.14 所示。传送 1000 个字节 CPU 所用时间为 1.202ms。因为中断响应需 2 个时钟周期，并且中断服务程序执行了 20 条指令后开始启动外设，所以外设每次中断请求的时间间隔为 $(2 + 20 \times 5) \times 1\text{ns} + 20000\text{ns} = 20102\text{ns}$，传送 1000 个字节所用时间为 $1000 \times 20102\text{ns} = 20.102\text{ms}$。因此，中断方式下 CPU 占用时间的

百分比约为 1.202/20.102＝5.98％。

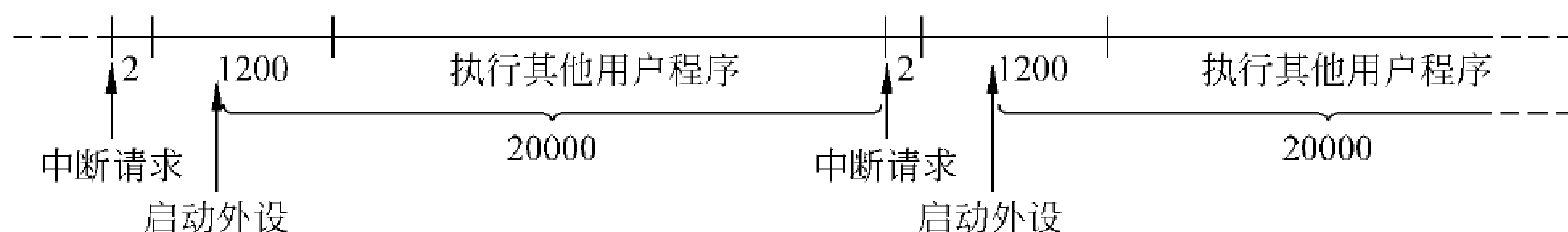


图 9.14　字符型设备中断方式下的 I/O 过程

(7) 根据对以上各种情况的分析，可以得到以下 4 个结论：①对于查询方式，定时查询比独占查询的 CPU 利用率高，但是，定时查询的时间间隔必须和外设的数据传输率和接口缓存情况相匹配。②对于快速设备，因为查询方式和中断方式的 CPU 开销大，使得 CPU 来不及处理外设传输的数据而导致数据丢失，因而快速设备不能采用这两种方式。③对于块传送设备，因为不需要每个字节都启动一次外设，所以，传送相同字节个数所用的时间比字符型设备的时间稍短。例如，对于 1000 字节的传送，独占查询方式下，块传送设备的时间是20.00104ms，而字符型设备是 20.02102ms；中断方式下，块传送设备的时间是 20ms，而字符型设备是 20.102ms。④外设速度越快，CPU 用于外设 I/O 操作的时间所占比例越大，因此，对于快速设备，应尽量减少 CPU 介入 I/O 的程度，即采用 DMA 方式更合适。

补充说明：在计算 CPU 占用时间百分比时也可用另一种方法计算得到。例如，对于(3)中断方式的计算，可以先求出 1 秒钟内该外设请求的中断次数为 1/(1B/50kB)＝50k，然后得到 1 秒钟内 CPU 用于数据 I/O 的时钟周期数为 $50k\times(2+1200)=6.01\times10^7$，因此在该设备传输过程中，CPU 用于该设备 I/O 操作的时间占整个 CPU 时间的百分比为 $6.01\times10^7/1G=6.01\%$。

参 考 文 献

[1] 袁春风.计算机组成与系统结构.北京：清华大学出版社，2010.

[2] Randal E. Bryant, David R. O'Hallaron. Computer Systems: A Programmer's Perspective. Prentice Hall, 2003.

[3] David A. Patterson, John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface, 3rd Ed. San Mateo, CA: Morgan Kaufman, 2004.

[4] John L. Hennessy, David A. Patterson. Computer Architecture: A Quantitative Approach, 3rd Ed. San Mateo, CA: Morgan Kaufman, 2002.

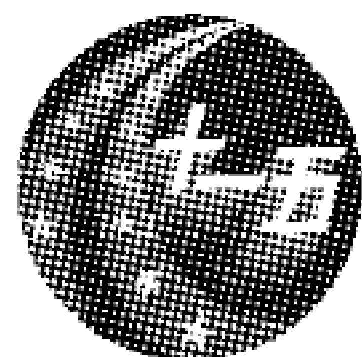
[5] Carl Hamacher, Zvonko Vranesic, Safwat Zaky. Computer Organization, 5E. The McGraw-Hill, 2002.

[6] William Stallings. Computer Organization and Architecture Design for Performance, 7th Ed. Prentice Hall, 2006.

[7] M. Morris Mano, Charies R. Kime. Logic and Computer Design Fundamentals, Third Edition. 北京：机械工业出版社，2008.

[8] Randal E. Bryant, David R. O'Hallaron 著，龚奕利，雷迎春译.深入理解计算机系统(修订版).北京：中国电力出版社，2004 .

[9] David A. Patterson, John L. Hennessy 著，郑纬民等译.计算机组成和设计 硬件/软件接口(第3版).北京：机械工业出版社，2007.



普通高等教育"十一五"国家级规划教材
21世纪大学本科计算机专业系列教材

近期出版书目

- 计算概论(第2版)
- 计算机导论(第2版)
- 程序设计导引及在线实践
- 程序设计基础(C语言)实验指导
- 程序设计基础(C语言)
- 程序设计基础(第2版)
- 程序设计基础习题解析与实验指导(第2版)
- 离散数学(第2版)
- 离散数学习题解答与学习指导(第2版)
- 数据结构与算法
- 算法设计与分析
- 计算机组成原理(第2版)
- 计算机组成原理教师用书(第2版)
- 计算机组成原理学习指导与习题解析(第2版)
- 计算机组成与系统结构
- 计算机组成与系统结构习题解答与教学指导
- 微型计算机系统与接口(第2版)
- 计算机操作系统(第2版)
- 计算机操作系统学习指导与习题解答(第2版)
- 数据库系统原理
- 编译原理
- 软件工程
- 计算机图形学
- 计算机网络(第2版)
- 计算机网络教师用书(第2版)
- 计算机网络实验指导书(第2版)
- 计算机网络习题集与习题解析(第2版)
- 计算机网络软件编程指导书
- 计算机网络习题解析与同步练习
- 人工智能
- 多媒体技术原理及应用(第2版)
- 形式语言与自动机理论(第2版)
- 形式语言与自动机理论教学参考书(第2版)
- 算法设计与分析(第2版)
- 算法设计与分析习题解答(第2版)
- C++程序设计(第2版)
- 面向对象程序设计(第2版)
- 计算机网络工程(第2版)
- 计算机网络工程实验教程
- 信息安全原理及应用